Carnegie Mellon University
Computer Science Academy

CMU CS Academy
CS1 Teacher Handbook
(2022-2023)

For changes to the CS1 curriculum for the 2022-2023 school year, please follow this link.

Table of Contents

Pedagogy

Course Description

Philosophy

Computer Science and computational problem solving are fundamental skills for engaging the 21st-century marketplace of ideas and economies. We believe that all students should have the opportunity to learn these skills as they will likely use them in whatever career they pursue.

Description

This free CS1 curriculum is designed for students in 9th grade with algebra readiness, though it is well-suited for students in 8th through 12th grades. No prior programming experience is required. It is inspired by 15-112, Fundamentals of Programming and Computer Science, a highly successful course taught at Carnegie Mellon University for the past 10+ years. It is predicated on the notion that learning about programming and computer science should be fun and engaging. This requires interesting problems to solve, as computational problem-solving is the core of computer science. It is why we utilize graphical problems in CS1—they are visually engaging, allow for multiple correct solutions, and provide visual cues when a solution goes awry, making debugging a cinch.

Details

There are 12 units in the course, which is split into two parts, CS1a (units 1-6) and CS1b (units 7-12) so that it can be taught as a year-long course or two semester-based courses. We firmly embrace learning-by-doing. Each unit provides interactive lecture notes with detailed examples, a set of engaging exercises to develop problem-solving skills, end-of-unit exercises that require students to synthesize the topics in that unit, and creative tasks, where students use their creativity and apply these topics in inventive ways.

Logistics

The course is entirely browser-based, including an Integrated Development Environment (IDE) that students use to create and run their programs. Using the IDE, students can write their code, watch their drawings appear on screen, and run our autograder to save you the hassle of grading—all without having to leave the website.

Commonly Asked Questions

What's the difference between CS and Programming?

Computer science is the study of theoretical, scientific, and mathematical approaches to information and computation, while programming is the hands-on application of designing, writing, testing, debugging, and maintaining computer software. A useful analogy is to think of computer science like architecture (design specs for a building that will make it beautiful, safe, and so on) while programming is like carpentry (the actual construction of those buildings).

Why is programming useful to all students?

Programming teaches problem-solving, creativity, and general computation skills.

Why is the course taught using Python?

Python is easy to read and understand and is popular in college and industry.

Why is this course online?

Keeping this course online eliminates any need for software installations by students or teachers, allows our curriculum to work with Chromebooks and other devices, and allows students to access their work from different computers.

Why is this course graphics-based?

Graphics are fun! Additionally, it allows students to experience the creativity in programming and actually see their bugs (which greatly aids debugging).

Why do students need algebra readiness?

CS1 requires algebra readiness for two reasons. The first and main reason is that programming requires abstraction. For example, students in algebra are used to the idea of a variable, which is an abstraction very similar to the notion of a variable in programming. The second reason is that the course sometimes uses algebra directly. The materials do not focus on math, but some algebra will show up from time to time, as it is a useful tool for graphics programming.

Advice on Running a Classroom

Our teachers come from a wide variety of backgrounds and experience levels, but that does not mean only some of them can be successful. We are here to help all of our teachers thrive not only with our curriculum but also with teaching computer science in general. To that end, we have compiled some suggestions and tips about how to use CS Academy in your classroom. This will undoubtedly be review for some of you, but hopefully others will find it helpful.

Using exercises

Differentiation

One of the first things you will notice about our site is how many exercises there are. We DO NOT recommend that students complete all of them. Instead, we assign points to each exercise based on its difficulty and then provide a suggested number of points each student should complete per section (found next to the star by each exercise section).

Our recommendation is just that: a recommendation. There are no built in requirements for activities, and we encourage you to grade and require exercises as you choose. This flexibility in required points and the varying difficulty of exercises allows for differentiation from student to student. You can encourage high achieving students to do more difficult problems (we don't recommend assigning problems, but more on that later) and you can require these students to earn more points than those who may be struggling with the material.

Collaboration

Creative tasks provide a great space for students to work collaboratively. Using the Groups tab in the Teacher Portal, you can assign students to groups in which multiple students can simultaneously edit the same code to make their task. In addition to this collaborative feature, the freeform nature of these assignments offers a unique opportunity for students to develop soft skills such as project management and teamwork.

Another great method for student collaboration is called pair programming. In this technique, two students work together at the same computer, one writing code and the other dictating what code to write. The student not typing is often referred to as the "head" since they come up with what code should be written, and the student typing is often called

the “hands” because they just type and bring up possible issues with what the head has told them. (Students should switch between these two roles quite often.)

Pair programming can be used with any of our exercises as you see fit and is a very common practice in industry, so students will be practicing an essential workplace skill. In the classroom, pair programming has been shown to keep students more engaged, increase retention, and help increase struggling students’ confidence.

Student Engagement

Another reason we provide so many exercise options is to allow students to pick activities they find interesting. (This is why we strongly recommend giving students the ability to choose which exercises they complete.) If students find their work interesting, they are more likely to stay engaged and put in more effort.

Advice for Teachers New to CS

Lead Learner

There is a learning curve to teaching computer science, and it is unrealistic to expect yourself to be an expert within a couple months. So, rather than aiming for expertise from the start, try to be a role model for how you want your students to learn the new content. For example, if a student asks a question you don’t know the answer to, admit it and then work together with the student to find the answer. And remember, we are here to help. If you have trouble finding an answer, just submit your question via the support tab and we will make sure to personally answer it.

Useful Resources

General CMU CS Academy Resources

CMU CS Academy Overview Slideshow
CMU CS Academy Overview Flyer

Website/Content Resources

Teacher Portal Guide
Console Message Guide
Graphics Reference

Teaching/Pedagogy Resources

Course Description and Standard Alignment
Alignment to State Standards
Scope and Sequencing and Pacing Guide
Creative Task Guide
Grading Creative Task Examples
Semester 1 Project Guide
Final Project Guide

Unit Review Videos

Unit 1: Creating Drawings
Unit 2: Functions, Mouse Events, and Properties
Unit 3: Mouse Motion Events, Key Events, and Methods
Unit 4: More Conditionals, Key Events, and Methods
Unit 5: Complex Conditionals and More Key Events
Unit 6: Groups, Step Events, and Motion
Unit 7: New Shapes, Local Variables, and For Loops
Unit 8: Math Functions, Random Values, and Nested Loops
Unit 9: (Optional) Types, Strings, and While Loops
Unit 10: Lists and Return Values
Unit 11: 2D Lists and Board Games
Unit 12: Final Project

Material by Unit

Unit 1: Creating Drawings

Unit Review

Shapes You Can Draw

Shapes with radius

- **Circle(centerX, centerY, radius)**
- **Star(centerX, centerY, radius, points)**
- **RegularPolygon(centerX, centerY, radius, points)**
    - Draws a shape with the specified number of points evenly spaced around the circle defined by (centerX, centerY) and radius.

Shapes with width/height

- **Rect(left, top, width, height)**
    - Rectangles are drawn from their left-top coordinate by default.
- **Oval(centerX, centerY, width, height)**

Miscellaneous

- **Line(x1, y1, x2, y2)**
- **Label(value, centerX, centerY)**
    - 'value' is the text displayed by the label.
- **Polygon(x1, y1, x2, y2, x3, y3, ...)**
    - Think of a Polygon like connect-the-dots. Each (x, y) pair is a dot. Lines are drawn between consecutive points, and the enclosed area is filled in.
        - There is no upper limit to the number of points in a polygon, but it will not be drawn unless there are at least three (having 0, 1, or 2 will not raise an error).

Optional Parameters

In addition to required parameters that determine position (x, y) and size (radius / width, height), shapes have optional parameters that you can use to modify their appearance, and a lot of them at that. For example, here's a rectangle with all optional values set to their default values:

```
Rect(left, top, width, height, fill='black', border=None,
```

```
borderWidth=2, opacity=100, rotateAngle=0, dashes=False,
align='left-top', visible=True)
```

List of Optional Parameters

Below is a complete list of the optional parameters in the course with notes about which shapes each can/cannot be used on:

fill—Specifies the color of a shape. 'black' by default. (All shapes)
border—A shape's enclosing border. None by default; can be set to any color. (All shapes except line)
borderWidth—How wide a shape's border is. 2 by default. (All shapes except line)
dashes—Whether a border has dashes. Can be set to a pair of values, (dashWidth, gapWidth), to create the dashes. False by default. (All shapes except labels)
opacity—How see-through a shape is on a scale of 0 (completely transparent) to 100 (completely opaque). 100 by default. (All shapes)
rotateAngle—The angle a shape is rotated in degrees where 0 is straight up and the angle increases clockwise. 0 by default. (All shapes)
align—Changes the orientation of the shape relative to the point put into the shape definition. For example, using "center" on a rectangle will draw the result centered on the input point rather than using that point as the top-left corner. This is the only property that CANNOT be changed after a shape is defined. Its default value changes based on the shape. (All shapes except polygons, Arcs, and Lines)
visible—A True/False value that determines if the shape is drawn or not. True by default. (All shapes)
roundness—How closely a star resembles a circle on a 0 to 100 scale. 0 produces a star that is as "spikey" as possible and 100 produces a star that is as "round" as possible. Around 57 by default. (Only used on stars)
size—Controls the font size of the text in a label. 12 by default. (Only used on labels)
font—Changes the font of the text in a label. Arial by default. (Only used on labels)
bold—Makes the text in a label bold. False by default. (Only used on labels)
italic—Makes the text in a label italicized. False by default. (Only used on labels)
lineWidth—Determines the thickness of a line. Must be greater than or equal to one. 2 by default. (Only used on lines)
arrowStart/arrowEnd—Determines if a line starts/ends with an arrow. This isn't introduced until Unit 7 but is included here for completeness of the list. Both are False by default. (Only used on lines)

Advanced Color Options

Rather than a solid color, fill can also be set equal to a gradient, aka one color transitioning gradually to one or more other colors, as seen in the masterpiece below (Note that you can change the orientation of the gradient).

While we certainly have a lot of named colors, computers can express many many more (16 million actually). To use these other colors, you'll need to change the fill of an object to be something called an RGB value, like in the example below. As you can see, an rgb value is composed of three colors, and while this is not necessary for students to understand, it may be helpful to know that all values must be between 0 and 255 and the first value represents the intensity of red, the second green, and the third blue (hence rgb).

- For reference, rgb(255, 0, 0) is pure red, rgb(0, 255, 0) is pure green, rgb(0, 0, 255) is pure blue, rgb(0, 0, 0) is black, and rgb(255, 255, 255) is white.

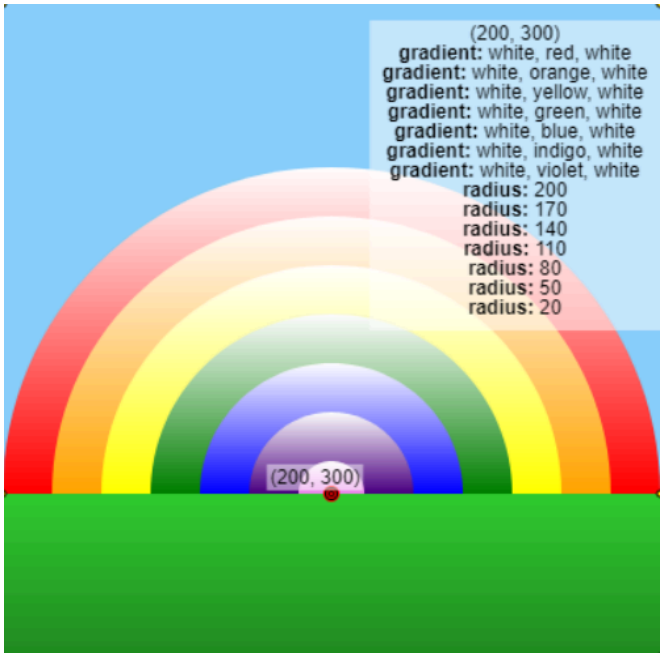
The Inspector

Once something has been drawn on the canvas, hover your mouse over some of the shapes. You will notice that some points are highlighted and that there is a grey box in the top-right corner with information in it, as shown in the image above. This is called the inspector and is vital to CMU CS Academy.

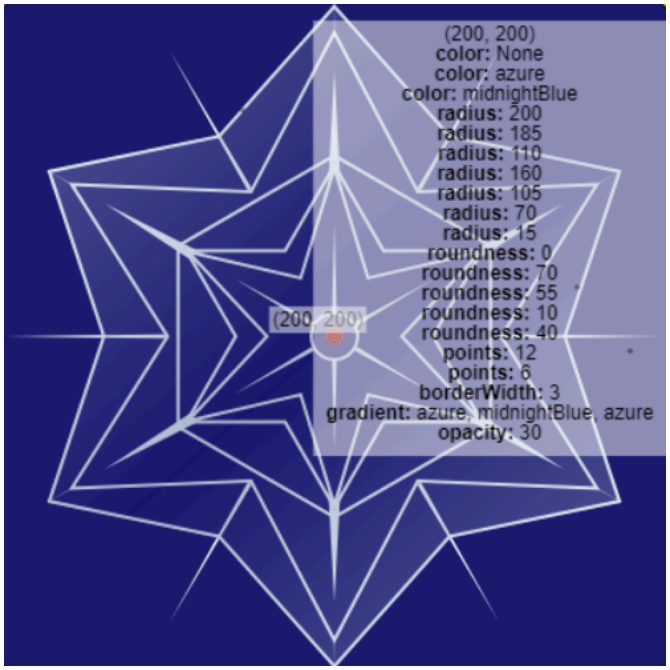
- Once we get further into the course and make interactive canvases, hold control to activate the inspector.

Hovering your mouse over one of the highlighted points will display information about any shapes at that point. Students can use the inspector on the solution view of the canvas to gather information about what to code. Even so, you will also notice that a lot is left up to the student to figure out on their own. The limited information we provide is meant to promote student problem solving skills and engage them rather than feed them answers.

- In the example below, the inspector tells us the x and y coordinates of the center of each rainbow layer, the radius of each, and the colors to be used, but it doesn't say which values correspond to which shapes.



The importance of the inspector cannot be overstated. For example, without the inspector, the below exercise would be nearly impossible to recreate:



Important Tips:

- All properties are listed by the inspector from the **back** of the canvas to the **front.**
    - So in the rainbow example, radius: 200 corresponds to the red circle.
- Make sure to pick **isolated points** when possible, i.e. points belonging to only one shape, so as to avoid confusion.

- E.g. in the picture below, hovering over the point (120, 290) gives you information about both the rock and its shadow, which might lead to confusion; (245, 240) only gives information about the rock.

Vocabulary

Shapes

- Rect(left, top, width, height)
- Oval(centerX, centerY, width, height)
- Circle(centerX, centerY, radius)
- Star(centerX, centerY, radius, points)
- RegularPolygon(centerX, centerY, radius, points)
- Polygon(10, 60, 50, 20, 30, 90)
- Line(x1, y1, x2, y2)
- Label('hello', centerX, centerY)

Colors

Go to the **Built-in Colors** page within the docs to view all of the colors you have to choose from.

- You can also make any other shade not listed using rgb(red, green, blue) values.

Common Optional Parameters:

- **fill:** The color of the inside of a shape.
- **opacity:** How opaque a shape is on a 0-100 scale.
- **border:** The color of a shape's border.
- **borderWidth:** The width of a shape's border, if it has one.

- **dashes:** Whether the border of a shape is dashed or not. False for no dashes, otherwise True or (dashWidth, gapWidth).

Troubleshooting/Advice

Exercise drawing looks correct

Drawings must be *exact* matches. Note that any solution in which this is accomplished will pass the autograder. For instance, an identical rectangle may be drawn with both the Rect function and the Polygon function, and the autograder will accept both! Since grading is based solely on the canvas, make sure to use the inspector to its fullest extent. It's your best friend! If you are getting an error, use the autograder and see if you have one of these common problems:

- Gradient directions are incorrect
- There is an incorrect number of shapes
    - Are you accidentally drawing extra shapes that are hidden?
    - Are you drawing redundant shapes?
        - i.e. drawing two identical shapes on top of each other
- Points don't match the solution
- Incorrect colors (spelling and/or rgb values)
- Incorrect property values
    - e.g. radius, border, etc

Nothing is happening

Check the console (area underneath the canvas) for an error message. If there is an error, look at that line number, as well as before and after to see if there are any syntax issues. If that doesn't fix the problem, make sure your code is not in comments.

Not sure where to start

Try drawing the image on paper, and translate your first steps into the code with the inspector.

Not sure how to get it to look right

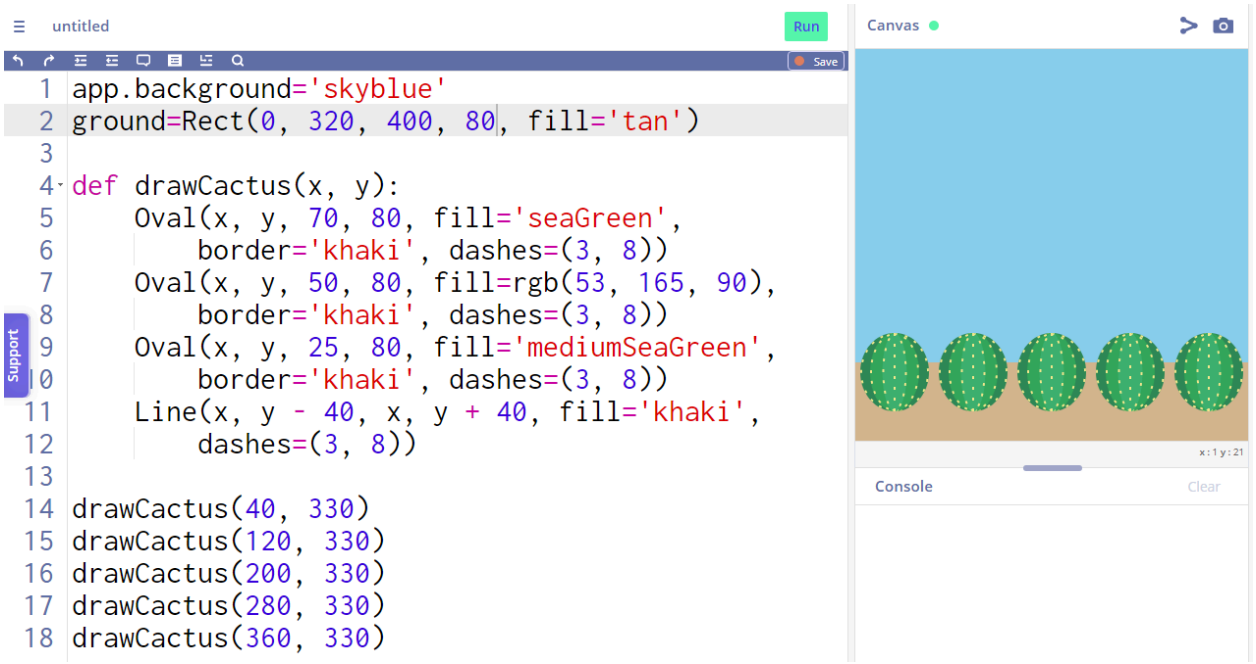
Layering is often used to cover up parts of shapes that aren't wanted. Code runs from top to bottom, so later lines of code get drawn on top of earlier ones.

Unit 2: Functions, Mouse Events, and Properties

Unit Review

Functions

A function is a piece of code that can take inputs and perform a certain task. Functions make performing repetitive tasks, like drawing collections of shapes, a whole lot quicker and easier. For example, look at the drawCactus function below:



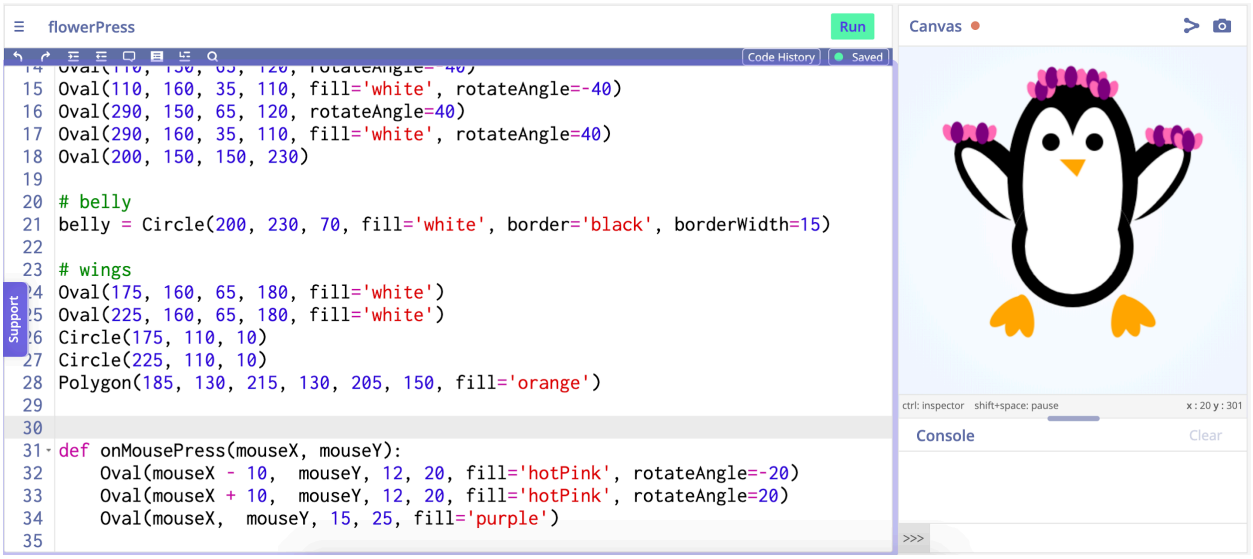

```
1  app.background='skyblue'
2  ground=Rect(0, 320, 400, 80, fill='tan')
3
4  def drawCactus(x, y):
5      Oval(x, y, 70, 80, fill='seaGreen',
6          border='khaki', dashes=(3, 8))
7      Oval(x, y, 50, 80, fill=rgb(53, 165, 90),
8          border='khaki', dashes=(3, 8))
9      Oval(x, y, 25, 80, fill='mediumSeaGreen',
10         border='khaki', dashes=(3, 8))
11     Line(x, y - 40, x, y + 40, fill='khaki',
12         dashes=(3, 8))
13
14 drawCactus(40, 330)
15 drawCactus(120, 330)
16 drawCactus(200, 330)
17 drawCactus(280, 330)
18 drawCactus(360, 330)
```

Mouse Events

You can click on the canvas to interact with it in interesting ways. The first two mouse events you'll learn are onMousePress(mouseX, mouseY) and onMouseRelease(mouseX, mouseY). onMousePress is called when the mouse is pressed, and onMouseRelease is called when the mouse is released (we pride ourselves on our original naming).

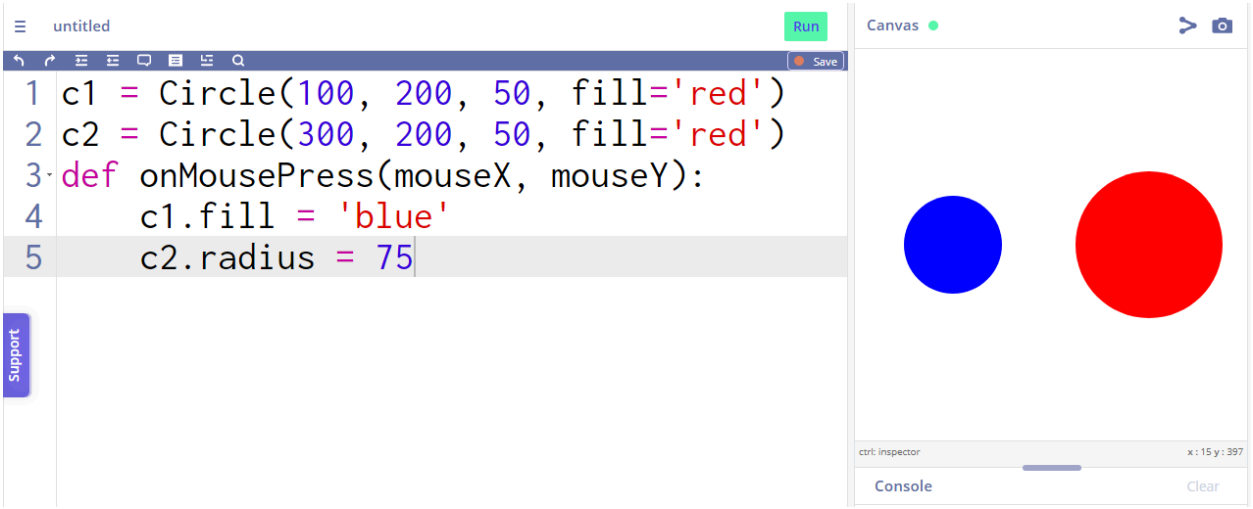
It is important to note that these two functions are special. They are part of a group of functions called event handlers. This means that we do not have to call these functions ourselves—the graphics package already does it for us. Whenever a certain event occurs, such as a mouse press or mouse release, the appropriate function is automatically called. Functions we write ourselves are different, because we have to call them in order for them to work.

Using these event handlers, let's make a function that draws a flower wherever we press the mouse (see image below):



Variables

You can store a shape in something called a variable. This is valuable if you want to change one of its properties later. Below, we draw two identical circles at different locations on the x-axis. However, in onMousePress(...), we change their properties in different ways. More on properties below.



Properties

Every shape has properties, which determine its position, size, and appearance. We set these properties initially with the arguments we provide when we create the shape, but we can also change them later on.

- For example, rectangles have the property of height. In the call Rect(0, 0, 10, 200), 200 is the ***argument*** that specifies the ***property*** of height.
- The distinction isn't that important, so don't get caught up in it. Just know how to use them. We're CS Academy, not Semantics Academy.

List of Properties

Size Properties: width, height, radius

Position Properties: left, right, top, bottom, align

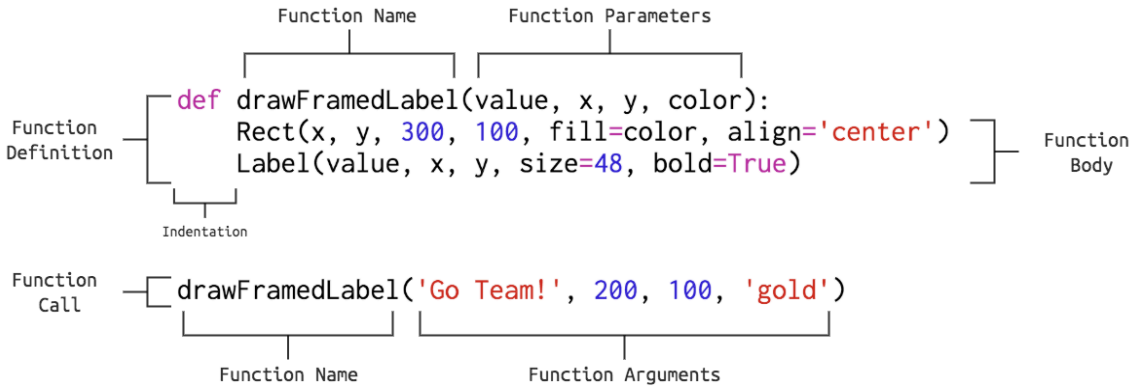
Fill Properties: fill, opacity

Border Properties: border, borderWidth, dashes

Other: visible (determines whether a shape is visible or not. Must be either True or False), rotateAngle (Determines the degree to which a shape is rotated. Can be any number, positive or negative, including decimal numbers)

Vocabulary

Functions



Mouse Events:

The onMousePress(mouseX, mouseY) function is called every time the canvas is clicked.
The onMouseRelease(mouseX, mouseY) function is called every time the mouse is released.

Properties

Position Properties: left , right , top , bottom , centerX , centerY
Size Properties: width , height
Alignment Properties: align

- Cannot be changed after the shape is initialized

Fill and Border Properties: fill, border, borderWidth, opacity, dashes
Other Properties: rotateAngle, visible

Shape-specific Properties

Circle Properties: radius
RegularPolygon Properties: radius , points
Star Properties: radius , points , roundness
Line Properties: x1 , y1 , x2 , y2 , dashes (False), lineWidth (2)
Label Properties: value , font ('arial'), size (12), bold (False), italic (False)

Miscellaneous

App.background: A property that determines the fill (color) of the canvas's background.

Global Variable: A variable that is declared outside any function that can be used at any point either inside or outside functions.
Boolean: A value that is either True or False.

Troubleshooting/Advice

Error saying a function/variable is undefined

Whenever you write code with functions or variables, you need to make sure you have defined them and are spelling them correctly so Python knows what you are talking about (spelling mistakes are a huge source of bugs).

Note that you must always define functions before using them (this is not necessarily required for variables).

Errors at function calls

Check to make sure that the function is spelled correctly and that your arguments are in the correct order.

Errors at function definitions

Check that you have a colon at the end of the line beginning with def and that everything is indented correctly. Also, make sure there is something in the function body: either productive code or “pass.”

Nothing is happening when the mouse is clicked/released

Make sure that you have spelled onMousePress/onMouseRelease correctly.

My mouse press/release events aren’t working as expected

Make sure that they are spelled correctly, onMousePress/onMouseRelease haven’t been mixed up, and that you use mouseX and mouseY correctly.

My properties aren’t changing as expected

Check your spelling! Python will give an undefined error when function names are misspelled, but it will not do the same for properties.

Carnegie Mellon University
Computer Science Academy

Unit 3: Mouse Motion Events, Conditionals, and Helper Functions

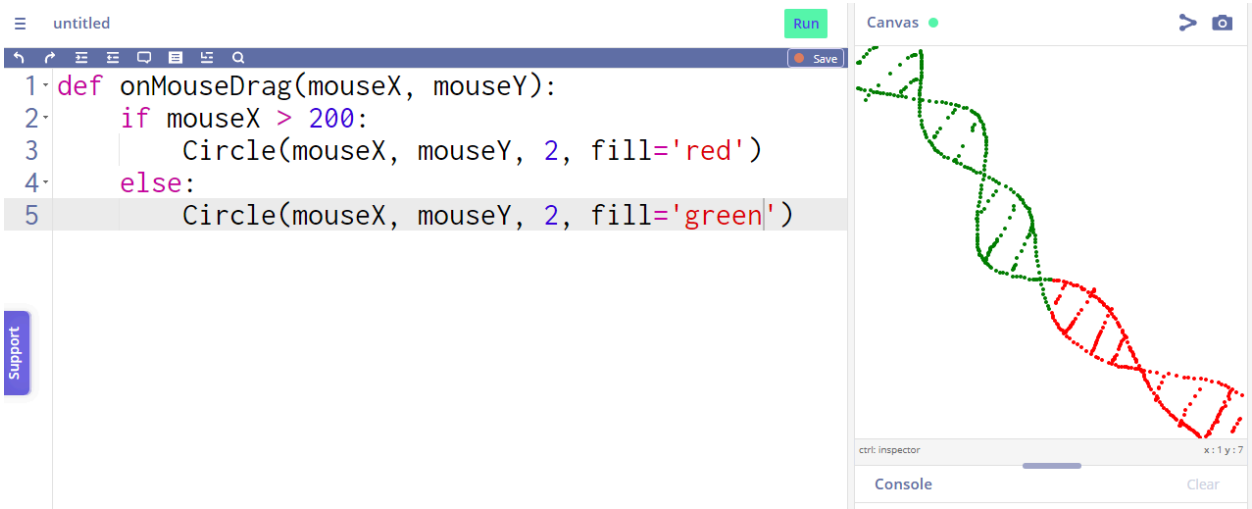
Unit Review

Mouse Motion Events

In addition to onMousePress and onMouseRelease, we have onMouseMove and onMouseDrag. Just like onMousePress and onMouseRelease, these names are pretty self explanatory. Moving your mouse over the canvas will trigger onMouseMove(...); clicking, holding down, and dragging will trigger onMouseDrag(...).

Conditionals

Any code contained within an if statement will only run when the condition is satisfied (in other words, it evaluates to True). Consider the masterpiece below:



Conceptually, conditionals are quite simple, but in practice the logic can get very tricky and easily tangled. Don't worry—they get easier with extensive practice.

Helper functions

A function that you call within another function is called a helper function. It behaves the same as any other function, so there's nothing new to learn here. Helper functions are used to keep code neat inside a particularly long function. For example, let's say you want to draw a pizza on the canvas. You will need to draw the crust, the sauce, and a variety of toppings. To keep your pizza-drawing function from becoming too lengthy and difficult to read, you could make a helper function for the crust & sauce, as well as helper functions for each type of topping.

Vocabulary

Conditional: An if/else statement that evaluates to either True or False; **if** it is True, the indented code chunk below runs.
Helper Function: A function that is called within another function. Largely helps to keep lengthy code neat.
onMouseMove(mouseX, mouseY): Called every time the mouse is moved and not pressed.
onMouseDrag(mouseX, mouseY): Called every time the mouse is pressed and moved.

Troubleshooting/Advice

Syntax error at equal signs

Make sure you use == and = correctly. == *compares* to values to check if they are equal, while = *sets* the variable on the left equal to the value on the right.

My conditionals don't work as expected

Check that your boolean expressions are correct, verify that your code is indented correctly, and make sure that you use else when you want events to be exclusive.

My code is hard to read

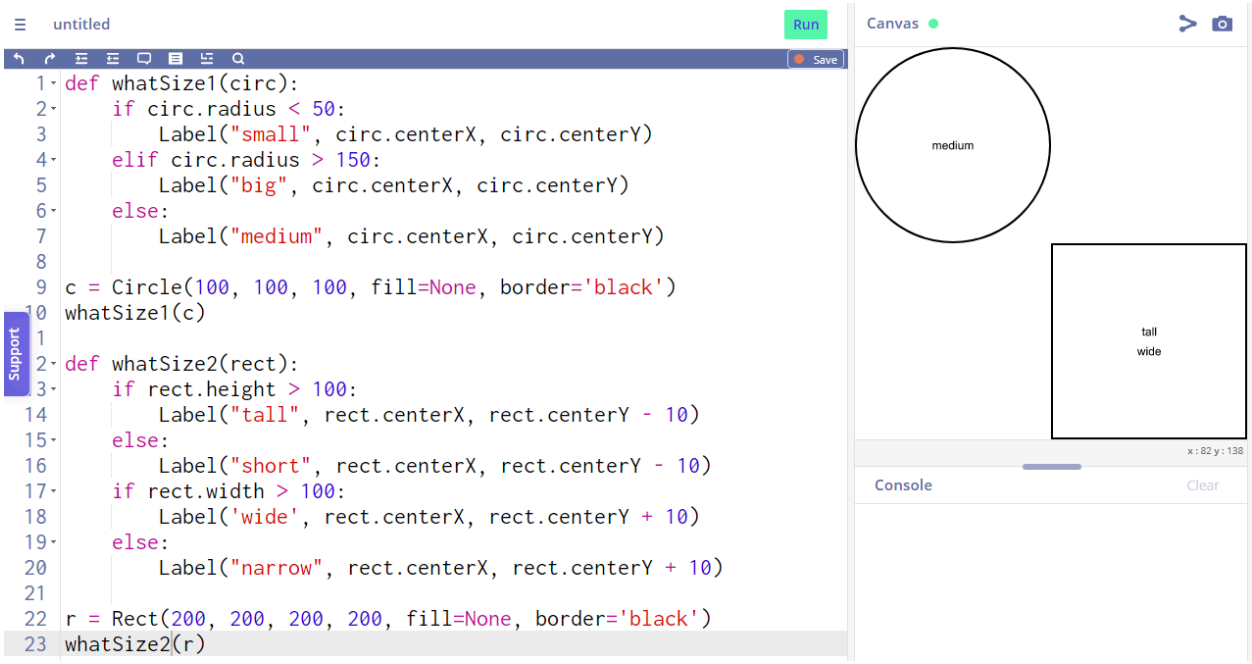
Make sure to use helper functions whenever you have code that is repeated. You should also use them to break large, complex functions into many smaller, less complex ones.

Unit 4: More Conditionals, Key Events, and Methods

Unit Review

If-elif-else, vs. multiple if's

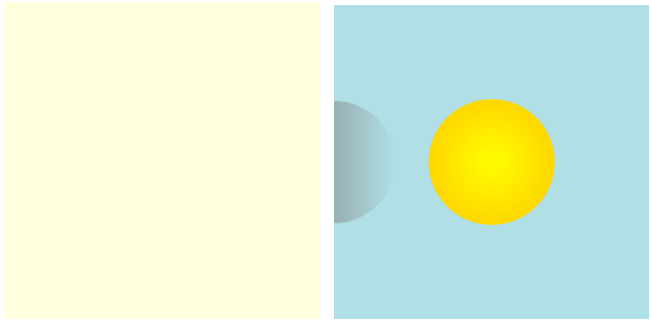
Think of each if statement as a door. When the condition you pass to the if statement is true, you're allowed to open it. Say you have 3 "if" doors. Say you pass the first if statement. Then you can open the first door, go inside, and crucially, *you can go back outside and check the other doors.* This means that you should only use multiple ifs in a row if you are checking conditions that are not mutually exclusive. Now say you have an "if" door, an "elif" door, and an "else" door. Again, let's imagine that you pass the first if statement. Then as soon as you open the door and go inside, *the door is locked behind you; you cannot evaluate any of the elif statements.* As you can see, elifs are valuable when you are checking conditions that are mutually exclusive.




```
def whatSize1(circ):
    if circ.radius < 50:
        Label("small", circ.centerX, circ.centerY)
    elif circ.radius > 150:
        Label("big", circ.centerX, circ.centerY)
    else:
        Label("medium", circ.centerX, circ.centerY)

c = Circle(100, 100, 100, fill=None, border='black')
whatSize1(c)

def whatSize2(rect):
    if rect.height > 100:
        Label("tall", rect.centerX, rect.centerY - 10)
    else:
        Label("short", rect.centerX, rect.centerY - 10)
    if rect.width > 100:
        Label('wide', rect.centerX, rect.centerY + 10)
    else:
        Label("narrow", rect.centerX, rect.centerY + 10)

r = Rect(200, 200, 200, 200, fill=None, border='black')
whatSize2(r)
```

Key Events

Just as you can use your mouse to interact with the canvas, you can also use your keys. Common examples include typing to fill in a label and using the arrow keys/ wasd keys to move a shape around.



The 2 key event methods taught in this unit are onKeyPress(key) and onKeyRelease(key). As you may have suspected, onKeyPress(key) is called when a key is pressed and onKeyRelease(key) is called when a key is released.

Shape Methods

For a single shape:

- **addPoint(x, y)**: Adds the point to a polygon.
- **toFront()**: Moves shape to front of canvas; i.e., draws it on top of any other shapes in the way.
- **toBack()**: Moves shape to the back of canvas, behind others
- **shape.hits(x, y)**: Returns True or False depending on whether (x, y) rests within a drawn part of the shape.
- **shape.contains(x, y):** Incredibly similar to shape.hits(x, y). The only difference is that if a shape's fill is None and you move the mouse within its boundaries, shape.hits(x, y) returns False, while shape.contains(x, y) returns True.

For multiple shapes:

- **shape1.hitsShape(shape2)**: Returns True if shape1 makes contact with shape2.
- **shape1.containsShape(shape2)**: Returns True if shape1 entirely encompasses shape2.

Shape Methods

Similar to functions. They perform some 'function' regarding an existing shape on the canvas. Unlike functions, however, they are called with the syntax 'shape.method()'.

shape.toFront(): Brings a shape in front of all other shapes on the canvas

shape.toBack(): Pushes a shape behind all other shapes on the canvas

shape.addPoint(x, y): Takes the x and y values of some point on the canvas and adds the point (x, y) to the shape

- *Only* works for polygons

shape.contains(x, y): Takes two values, x and y, and simply tells us whether or not the shape contains the point (x,y). Returns True or False

shape.hits(x, y): Only different from contains in one way—if the shape's fill is None, any point inside the shape will return False

shape1.hitsShape(shape2) Tests if a shape hits another shape

shape1.containsShape(shape2) Tests if the first shape fully contains the second shape

app.stop(): Called when we want no more events happen

- For example, calling app.stop() would prevent any further onKeyPress(key) calls from having any effect.

Miscellaneous

If-elif-else: A series of if/ elif/ else statements that ceases evaluating once any one of them evaluates to True; only the first true statement in the chain will run
onKeyPress(key): Called every time a key on the keyboard is pressed
onKeyRelease(key): Called once a key that has been pressed is released
Custom Properties: Just as shapes have properties like fill, border, and opacity, you can give a shape its own new property. Set it by calling shape.property = [blank]

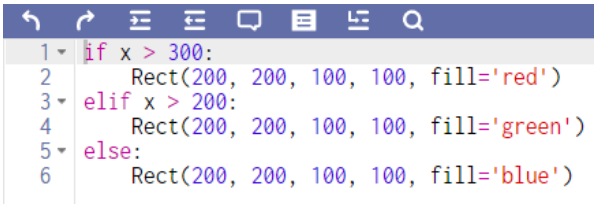
- For example, **circle.colorIfPressed = 'Yellow'** would store yellow as a property of the shape. Then, you could easily set **circle.fill = circle.colorIfPressed** in onMousePress(mouseX, mouseY).

Troubleshooting/Advice

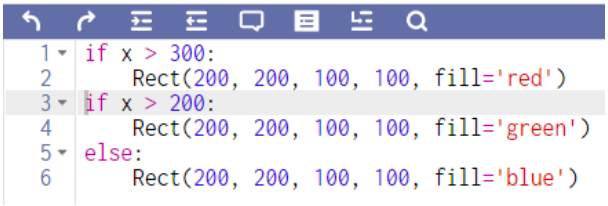
My conditionals still don't work

Check all of the suggestions for conditionals in unit 3, and make sure to use elif statements when you need to. Inexperienced coders sometimes think elif isn't useful, but in certain situations, it is absolutely necessary.

- For example, these two chunks of code give different outputs when x > 300:



```
1  if x > 300:
2      Rect(200, 200, 100, 100, fill='red')
3  elif x > 200:
4      Rect(200, 200, 100, 100, fill='green')
5  else:
6      Rect(200, 200, 100, 100, fill='blue')
```



```
1  if x > 300:
2      Rect(200, 200, 100, 100, fill='red')
3  if x > 200:
4      Rect(200, 200, 100, 100, fill='green')
5  else:
6      Rect(200, 200, 100, 100, fill='blue')
```

Custom properties aren't modified as intended

Make sure you have spelled your properties correctly; otherwise Python just makes the misspelling a new property (this is the reason misspelled properties don't give error messages, as we mentioned in unit 2).

Unit 5: Complex Conditionals and More Key Events

Unit Review

Complex Conditionals

Complex conditionals are conditionals that can be broken down into several simpler tests. Compound conditionals contain the keywords `and`, `or`, or `not` .|. Nested conditionals are conditionals that contain other conditionals inside of them (nested).

- **And:** Can be used to test if two conditionals are both true.
    - For example, if ((isSummer == True) and (isSunny == True)) tests if it is both summer and sunny outside.
- **Or:** Can be used to check if at least one of two conditionals is true.
    - For example, if ((isSummer == True) or (isSunny == True)) tests if it is sunny outside or it is summer
        - If both conditions are True, the result is still True.
- **Not:** Not makes a conditional the opposite of what it normally means. So, (not True) = False and (not False) = True.
    - This can be very useful when you want to check if something is False. For example, if you want to see if it is not sunny outside, you could check if ((not isSunny) == True)

Nested and compound conditionals are mostly interchangeable, it's just a matter of whichever makes your code the most clear and concise. If you have a ton of nested conditionals, condensing some of them into compounds might be beneficial, and vice versa. This is ***not*** a strict rule, but in 15-112 at CMU, the following style guidelines tend to be helpful for beginners in keeping code neat. Consider the following criteria when trying to clean up functions with lots of complex conditionals:

	Function is short (<20 lines)	Function is long (>20 lines)
Lines are short (<80 characters)	Nice!	Use compound conditionals
Lines are long (>80 characters)	Use nested conditionals	Write helper function(s)

More Key Events

In addition to the other key events discussed earlier, we have **onKeyHold(keys**). This function is slightly different than the others. First, as the name suggests, it is called when a key is held down—between onKeyPress(key) and onKeyRelease(key). It is unique up to this point in the course in that it is called repeatedly as long as the key is held down. In addition, this function has the parameter keys rather than key. meaning that it uses something called a list. We will discuss lists more in the future; for now, just know that the keys list can store any number of keys that are being held down at once. To check if a key is being held, use a test with the keyword "in."

- For example, to see if 'x' is being held down, you would use
  if ('x' in keys).

Vocabulary

Conditionals

Complex Conditionals: Conditionals that can be broken down into several simpler tests.

Compound Conditionals: Whenever you use `and`, `or`, or `not` to evaluate complex conditionals.

Nested Conditionals: Whenever you put simple conditionals inside of one another to evaluate complex conditionals.

Key Events

onKeyHold(keys): Is called repeatedly as long as a key—or a combination of keys—is held down (between onKeyPress and onKeyRelease)

Should I use nested conditionals or “and”/”or”

See the chart on the previous page.

My conditionals don’t work

Start by checking all of the suggestions from units 3 and 4, as those errors are most pervasive. Additionally, check that all indentation is correct when working with nested conditionals and that boolean values are grouped correctly when working with compound conditionals.

- Always place each condition in its own set of parentheses.

I’m pressing the right key, but onKeyHold isn’t working

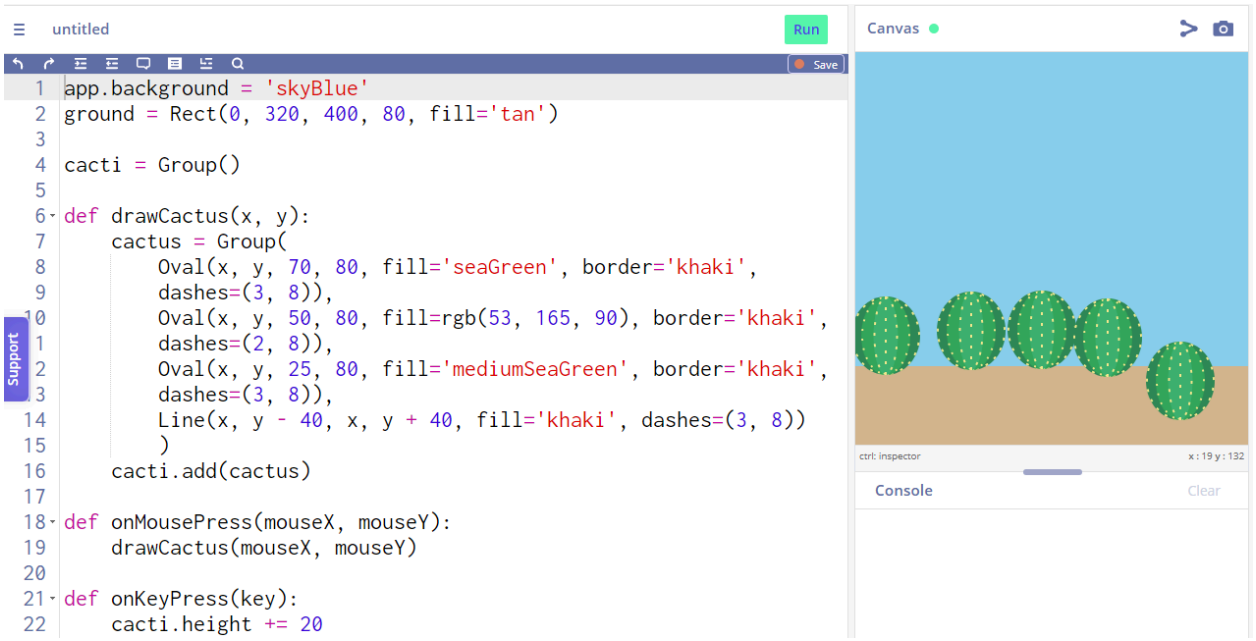
Make sure that you are checking that the key is in keys, not that it is equal to it like in onKeyPress. Also, any value in keys is case sensitive and changes if you hold control, alt, or any other modifying key. Make sure none of these special keys are pressed when testing.

Unit 6: Groups, Step Events, and Motion

Unit Review

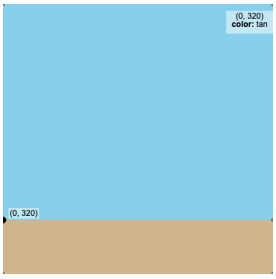
Groups

Groups are great for... well, grouping multiple shapes together. Let's revisit the cactus code from earlier. Now, with each key press we increase the height of the whole group of cacti:




```
app.background = 'skyBlue'
ground = Rect(0, 320, 400, 80, fill='tan')

cacti = Group()

def drawCactus(x, y):
    cactus = Group(
        Oval(x, y, 70, 80, fill='seaGreen', border='khaki',
        dashes=(3, 8)),
        Oval(x, y, 50, 80, fill=rgb(53, 165, 90), border='khaki',
        dashes=(2, 8)),
        Oval(x, y, 25, 80, fill='mediumSeaGreen', border='khaki',
        dashes=(3, 8)),
        Line(x, y - 40, x, y + 40, fill='khaki', dashes=(3, 8))
        )
    cacti.add(cactus)

def onMousePress(mouseX, mouseY):
    drawCactus(mouseX, mouseY)

def onKeyPress(key):
    cacti.height += 20
```

Why might you want to put all the cacti into a group? Because there are a lot of handy group methods that you can do cool stuff with, and it's a lot more efficient than changing each shape individually.



Group Properties/Methods

Groups have the exact same properties that all shapes share (this does not include shape specific properties like radius). Similarly, methods that work on all shapes work on groups but change every shape within the group. This is incredibly useful and makes coding complex behaviors much easier. In addition, there are some group specific methods, which are listed below:

group.add(shape**):** Adds a new shape to the group

group.remove(shape): If you give a shape a variable name, say s, you can later remove it from the group by calling group.remove(s).

- This will also delete the shape from the canvas.

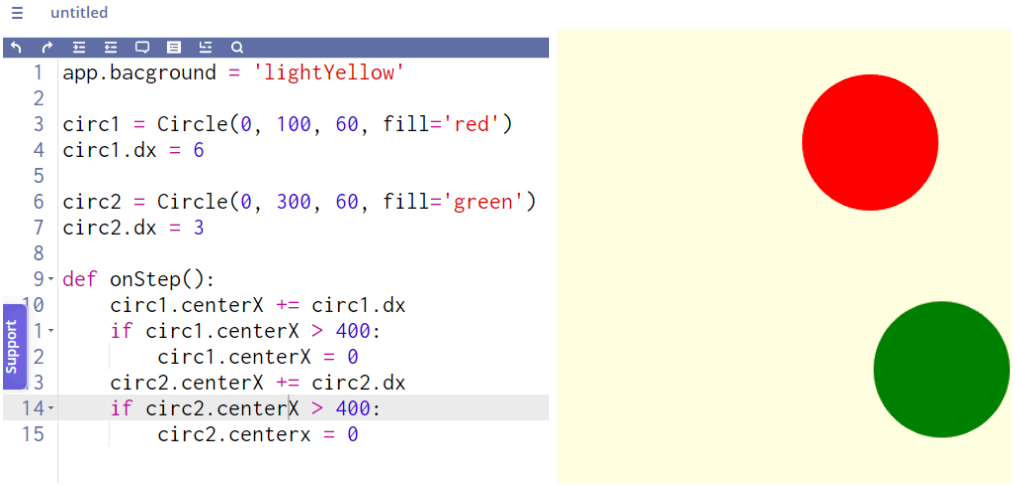
group.clear(): Removes all shapes currently in a group

- Just like remove, this deletes every shape from the canvas.

Step Events and Motion

For the first time in our course, we will now be dealing with events that are independent of the user. The function onStep() is called repeatedly a certain number of times per second. This is similar to onKeyHold(keys), but rather than being called over and over again only when the user is holding down a key, onStep() is always called over and over again. It is an incredibly useful event function that perhaps most notably allows you to simulate smooth, constant motion on the canvas. The number of times this function is called per second can be changed by setting app.stepsPerSecond to the desired number of calls per second.

onStep() is commonly used to create motion, like in the example below. However, it can also be used to do many other things including changing rotateAngle, increasing size, etc:really changing any shape property in a formulaic way.



Groups

Group: A collection of shapes that can all be treated as one.

Group Properties:

- Properties Shared with Shapes
  - left, right, top, bottom, centerX, centerY, opacity, fill, rotateAngle, visible, width, height
- Group Specific Properties
  - **Group.children:** Stores all of the shapes in the group as a list

Group Methods:

- Methods Shared with Shapes
  - group.toFront(), group.toBack(), group.hits(x, y), group.contains(x, y), group.hitsShape(shape), group.containsShape(shape)
- Group Specific Methods
  - **group.add(shape):** Adds shape to the group
  - **group.remove(shape):** Removes shape from the group
  - **group.clear():** Clears all shapes from the group

app.group: A group in which the app stores all shapes on the canvas

Step Events

onStep(): Called repeatedly as time passes

onSteps(n): Calls onStep n times in a row

- Used for autograding and not of much use when coding

app.stepsPerSecond: Sets how many times per second onStep() is called

I get an error modifying my group's properties

Make sure that you are modifying properties that all shapes share. For example, you can modify group.centerX, but you can't modify group.radius. Also check that your modifications don't make any property an illegal value. For example, if you decrease group.opacity, and one shape's opacity goes below zero, you will get an error.

My motion animation doesn't work correctly

Check that you are updating dx and dy correctly. A common error is to change the sign of dy and dx and also switch between addition and subtraction. Either one of these will change motion and doing both will cause strange behavior.

If your problem deals with bounded/wraparound motion, make sure you are using the correct shape property to bound it as you intended, your conditionals use the correct values for the canvas edges, and your adjustment to the shape places it where you intended.

Carnegie Mellon University
Computer Science Academy

Unit 7: New Shapes, Local Variables, and For Loops

Unit Review

New Shapes

Arc(centerX, centerY, width, height, startAngle, sweepAngle) effectively draws an oval but only fills in the portion starting at startAngle and moving clockwise by [sweepAngle] degrees.

Additionally, we can now make arrows by setting the optional parameters arrowStart/arrowEnd to True when drawing a line.

Local Variables

We have been using variables to store shapes for many units now, but they can store anything you want them to, not just shapes. So far, we have frequently been creating variables outside of any function, and then modifying them inside of other functions. This type of variable—one defined outside of any function,— is called a global variable, and it can be accessed and modified anywhere in the code.

- Global variables (often just called globals) should be used very sparingly.

We have also been using function parameters extensively, and one of their defining properties is that they can't be referenced outside of their function. (For example, it makes no sense to reference keys in onMousePress). We haven't focused a lot on this behavior, but parameters behave this way because they are a type of local variable. A local variable is

a variable defined inside of a function, which it can only be referenced inside of its function. (It is “invisible” anywhere outside of its function.)

While parameters were used to introduce the idea of local variables, they are not the only type of local variables. In fact, the more common type is a helper variable, which is a variable defined inside of a function in order to make reading and writing the code easier.

In the example above, width and height are global variables, x, y, and text are parameters, and bw and s are helper variables.

- Note—this is bad code. There is no need to make width and height globals. It is done here as a demonstration, but in practice you should avoid using globals wherever possible.

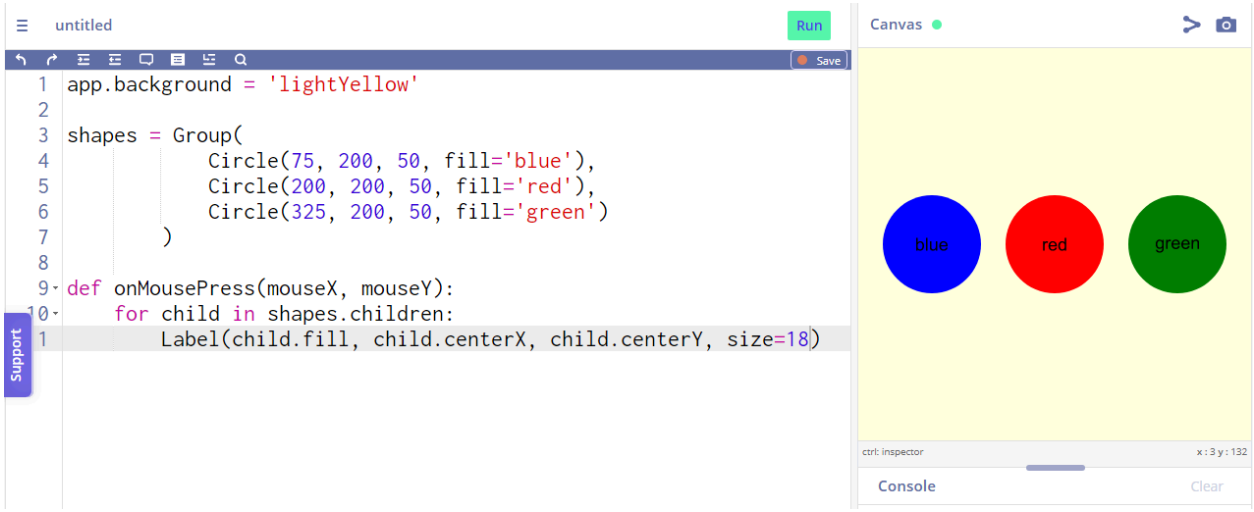
For Loops:

Loops are perhaps one of the most important concepts in programming. To understand how they work, imagine that you are flipping through a book. On each page, you do something (read it). Thinking in Python, we might say:

```
for page in book:
  read(page)
```

Generally, we don’t read books in Python (besides dictionaries). More often, you will be looping through numbers, where you can think of each number as a page. We use range(n) to represent the numbers (0, 1, 2,... n-1) like in the example below.

Even if you're already familiar with loops, there's still one use particular to our framework that you should know: looping through a group. If you have a group, g, comprised of shapes s1, s2,... You can loop through the group's 'children', g.children, to change the properties of each shape in the group. This behavior is illustrated in the example below:



Vocabulary

New Shapes/Properties

Arc(centerX, centerY, width, height, startAngle, sweepAngle)

Arrow → Line(x1, y1, x2, y2, arrowStart=True, arrowEnd=True)

arc.startAngle: The angle of the beginning of the arc

arc.sweepAngle: How much of the oval is drawn

arrow.arrowStart: Boolean that determines if there is a triangle at (x1, y1)

arrow.arrowEnd: Boolean that determines if there is a triangle at (x2, y2)

Variable Types

Global Variables: Variables defined outside of any functions that can be used anywhere in your code.

- Try to avoid using global variables

Local Variables: Variables defined inside a function that can only be accessed inside of that function.

Function Parameters: Local variables that represent the values passed into a function.

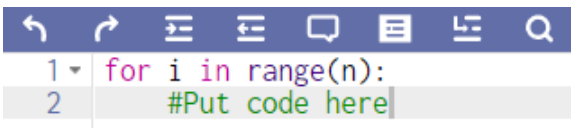
Helper Variables: Any local variable that is used to make code clearer and easier to write/read.

Loops

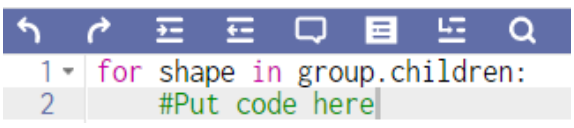
range(n): When used in a for loop, makes the loop run n times and increments the looping variable from 0 up to and not including n.

For Loop: A loop that runs for a predetermined amount of iterations.

- Exs. with range vs. groups



```
1 for i in range(n):
2     #Put code here
```



```
1 for shape in group.children:
2     #Put code here
```

My arc doesn't look right

Make sure that you didn't accidentally switch startAngle and sweepAngle, and remember that zero degrees is straight upward and the angle increases as you go clockwise.

Undeclared variable error

Make sure that you didn't misspell your variable name or reference a variable outside of its scope.

- i.e. Referencing a local variable from one function in another

My loop doesn't run the correct number of times

Make sure that the number inside range is the number of iterations you want.

Unit 8: Math Functions, Random Values, and Nested Loops

Unit Review

More Operators

As you have seen up to this point, computer science relies heavily on mathematics. Anything in code that represents a basic math function (like +, -, *, and /) is called an operator. You have been using operators for many units now, but there are many more that you have not been introduced to yet.

One such operator is called integer division and is denoted by // in Python. Integer division divides two numbers, and then ignores any decimals that are produced.

- For example, 5//2 = 2, 12//10 = 1, and 9//3 = 3

The other new operator introduced in this unit is remainder (sometimes called modulus) and is denoted by %. Remainder returns the remainder left over when you divide two numbers, just like you learned back in elementary school.

- For example, 5%2 = 1, 12%10 = 2, and 9%3 = 0

Remainder has many uses, but in this course, we will only be interested in two.

- If x%2 == 0, x is even and if x%2 == 1, x is odd.
- x%10 returns the ones digit of x.

Math Functions

In addition to these new operators, we will also be introducing some new math functions in this unit.
abs(x) - returns the absolute value of x
distance(x1, y1, x2, y2) - returns the distance between the two input points
angleTo(x1, y1, x2, y2) - returns the angle from one point to another with 0 degrees being straight up
getPointInDir(x1, y1, angle, distance) - returns the x and y coordinates of a point the given distance and angle from the input

Random values

We will often use random values in our exercises to make them change each time they run. While there are several ways to do this, the only one we will use in this course is randrange.

- **randrange(lo, high):** returns a random value between lo and high, including lo but excluding high. (So randrange(-1,2) returns -1, 0, or 1.)

Nested loops

Whenever we put one loop inside of another, we are using nested loops. This is useful for 2D grids, where the outer for loop represents rows and the inner one represents columns. The code below shows a simple example of this:

Vocabulary

More operators

Operator: Any predefined math function in code

- Ex. +, -, *, and /

Integer Division (//): Divides two numbers, but ignores any decimals

- Ex. 5//4 == 1 and 10//3 == 3

Remainder/Mod (%): Returns the remainder left after dividing two numbers

- Ex. 7%3 == 1 and 18%7 == 5 and 18%9 == 0

More Math Functions

abs(n): Returns the absolute value of a number

- Ex. abs(-8) == 8 and abs(9) == 9

distance(x1, y1, x2, y2): Returns the distance between two points

angleTo(x1, y1, x2, y2): Returns the angle between two points where 0 degrees is straight up

getPointInDir(x, y, angle, distance): Goes the specified distance from the input point (x, y) at the given angle and then returns the x and y coordinates (xOut, yOut)

range(lo, hi, step): Sets the iteration variable to lo and then increments it by steps of size step as long as it is less than hi. For example, range(1, 5, 2) would make the iteration variable 1 then 3, but it would never reach 5, because 5 is equal to hi. (If no step is provided it defaults to one; similarly, if no lo is provided, it defaults to zero.)

randrange(low, high): Returns a random number from low up to but not including high

Troubleshooting/Advice

My for loop/randrange isn't using the correct range

Make sure that you put the correct numbers into range. This can be confusing since the lower number is included in the range of possible values while the larger number is not.

- e.g. randrange(10, 15) will only ever return 10, 11, 12, 13, or 14. Similarly, for i in range(10, 15) will loop 5 times, setting i equal to 10, 11, 12, 13, and then 14.
    - In for loops, make sure that your step size is correct

My function looks correct and I am using randrange, but I can't pass the autograder

Check the comments in the code to make sure you used randrange in the correct spot, and if there are multiple calls to randrange, make sure they are in the correct order. While the order of randrange doesn't do anything from a logical standpoint, it can cause problems with our autograder.

Unit 9 (Optional): Types, Strings, and While Loops

Unit Review

Types

We don't cover types too extensively, but the following chart gives a good summary:

integers (or **ints**)	Numbers such as 42, 0, and -5
floats	Numbers with a decimal point, such as 1.2, -5.8, and 3.0
strings	Characters within single or double quotes, such as 'hello' and "this works, too!"
booleans	One of two values: `True` or `False`
shapes	Rectangles, ovals, circles, labels, ...

Note that you can combine 2 strings using the plus sign, you can turn an int into a string using str(), and you can turn a string into an int using int().

- "Hello" + ", World!" is equal to "Hello, World!"
- str(42) is equal to "42"
- int("42") is equal to 42

Why don't we cover types extensively? While types are incredibly important, they are out of the scope of this course, and arguably of Python itself. Python is a high-level language, meaning it is basically built upon a more primitive language and comes loaded with all sorts of helpful tools—tools that allow you to not sweat over types, among other things. But if you have any high fliers or are a bit of a mad-hatter yourself, consider the following rabbit holes:

- **Functional Programming**
  - **Immutability**
- **Imperative Programming**
  - **Size of Various Types in C Language**

Strings

As mentioned earlier, text in Python is more often referred to as strings. In the following examples, consider s = 'CS Academy'.

It is possible to look at specific letters in a string by referring to their location relative to? the beginning (starting at 0 like always). This is known as indexing. The following examples show the correct syntax.

- s[0] = 'C'
- S[1] = 'S'
- S[-1] = S[len(s) - 1] = 'y'

Just like you can look at an individual character in a string, you can also look at a specific series of characters. This is called a substring, and we can obtain it in a very similar way to indexing. The inputs for getting substrings are the same as those in range, except we use colons instead of commas to separate them.

- S[0:2] = 'CS'
  - Note that s[2] (the space) is *not* included
- S[3:] = 'Academy'
- S[:3] = 'CS '
- S[::] = 'CS Academy'
- S[::2] = 'C cdm'

Additionally, we can check if one string is contained in another by using in. For example, we can write:

```
if "CS " in "CS Academy":
    print("True")
```

Which will print "True" since "CS " is a substring of "CS Academy".

String Methods

Just like groups and shapes have their own specific methods, strings do too.

.isupper(): Returns a boolean value that tells if the input string is all uppercase.
.islower(): Returns a boolean value that tells if the input string is all lowercase.
.isalpha(): Returns a boolean value that tells if the input string is all letters.
.isdigit(): Returns a boolean value that tells if the input string is all digits.
.upper(): Returns a new string that is the same as the input but has all uppercase letters.
.lower(): Returns a new string that is the same as the input but has all lowercase letters.

While Loops

While loops are very similar to for loops, but they continue looping until some condition is no longer True. Consider the book analogy again:

```
page = 1
eyesTired = False
while (eyesTired == False):
  read(page)
  page += 1
  if (page > 100):
    eyesTired = True
```

While loops are used when we aren't sure how many times we have to repeat code. (If we did know, we would just use a for loop.)

Types

Type: The classification of a value

- Important examples include integers, floats, strings, booleans, and shapes

Type Error: An error that occurs when a value is an unexpected type

Strings

Concatenation: Combining two strings

- For example “this is “ + “concatenation” = “this is concatenation”

str(x): Turns its input, x, into a string

- For example, str(42) = “42”

app.getTextInput(“message”): Displays the input string (in this case “message”) as a prompt to the user and returns the response they give as a string

int(“x”): Turns the string of an integer into an integer value

- int(“42”) = 42

string.isupper(): Returns True or False if all characters in a string are uppercase

string.islower(): Returns True or False if all characters in a string are lowercase

string.isalpha(): Returns True or False if a string contains only letters

string.isdigit(): Returns True or False if a string contains only numbers

string.upper(): Returns a new string that is the same as the old one, but all of its characters are uppercase

string.lower(): Returns a new string that is the same as the old one, but all of its characters are lowercase

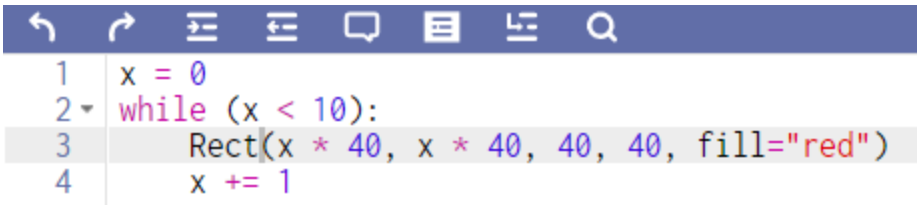
String Indexing: Accessing a certain character in a string based on its position

- Ex. “Random String”[3] = “d”

While Loops

While Loop: A loop that runs for as long as a condition is True rather than based on a predefined range.

- Ex.



```
1  x = 0
2  while (x < 10):
3      Rect(x * 40, x * 40, 40, 40, fill="red")
4      x += 1
```

I got a type error

A type error occurs when you accidentally use a value of the wrong type, so look through your code and try to figure out which values are an unexpected type. Type errors are very common between ints and strings, since a number like 42 can be an int or the string “42”. If this is the problem, use int() to turn a string into an int and str() to turn an int into a string.

Type errors are also common when you try to assign a specific string index to a new value. Unlike with lists, you cannot change a specific index of a string. Instead, you have to make a new string with the desired changes.

I got an out of bound error when indexing a string

Out of bound errors are very common when working with strings, and occur when you try to access a string index that does not exist within the string. Check to make sure all indexes are less than the string length (not equal to the string length since the indexes start at zero). If indexing inside of a loop, check the loop bounds to make sure you never use an index higher than the string’s length minus one.

My code never finishes running/my loop never seems to end/the console printed the message “Your code appeared to be in an infinite loop, so it was terminated”

All of these problems are caused by an infinite loop, which happens when the condition used to break out of a loop never becomes False. The most general way to fix this issue is to check your looping condition and figure out why it isn’t becoming False. Common causes include incorrectly incrementing variables and using a boolean to break out of the loop but never changing its value within the loop.

Unit 10: Lists and Return Values

Unit Review

Lists

Lists, like groups, are a collection of various elements. However, while groups exclusively store shapes, lists can store anything.

The syntax for making a list in Python is lst = [... things you want to store ...] (i.e. two enclosing brackets). The elements in a list are separated by commas. So a valid list might be lst = [1, 2, 3, “easy”, “as”].

Another important distinction between lists and groups is that lists can be indexed, just like strings, but groups can’t.

Despite their similarities, it’s generally wiser to put shapes in a group than a list. Lists do not have access to the invaluable functions & properties that groups do. In fact, the only reason you would ever want to use a list instead of a group to store shapes is if you need the index of the shape; you cannot index in a group.

List Methods:

- **L.append(elm):** Adds an element to the end of a list
- **L.pop():** Removes *and returns* the element from the very end of list
    - Optionally, you can instead call L.pop(index) to remove & return the element stored at [index] from the list
- **choice(L)**: Returns a randomly-chosen element from the list

Return Values

group.hitTest(mouseX, mouseY): Returns the frontmost shape within the given group that is hit by the input x and y coordinates.

To return something at the end of a function, simply call “return something”:

```
def isEven(num):
  if num % 2 == 0:
   return True
  else:
   return False
```

Note that a function can only return one thing; once it returns, any further code is not evaluated. So the code above can be rewritten like so:

```
def isEven(num):
    if num % 2 == 0:
        return True
    return False
```

We can make this code even cleaner still! You can actually return logical statements:

```
def isEven(num):
    return (num % 2 == 0)
```

Lists

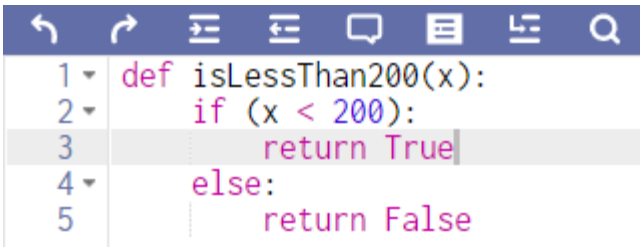
List: A data type that stores a collection of values in a specific order.
List Methods

- **list.append(x):** Adds a new value to the end of a given list.
- **list.pop(index):** Removes the value of a list at the given index and returns it (Defaults to removing the last value of the list).
- **choice(list):** Returns a random value from the list.

Functions that Return Values

group.hitTest(x, y): Checks to see if any shape in the given group is hit at the point (x, y) and, if so, returns that shape. If there is no shape at that point, it returns None.
return: When called in a function, immediately ends the function and returns the specified value.

- Ex.



```
1 def isLessThan200(x):
2     if (x < 200):
3         return True
4     else:
5         return False
```

I got an out of bound error when indexing into a list

List indexing and string indexing work in the same way, so just like with strings, whenever you try to access an index greater than or equal to the length of the list (due to zero indexing), you get an error.

Check that your indexes are smaller than the length of your list and, if you are using a loop, that the conditional doesn't let your index get greater than or equal to the list length. Since lists *can* be changed by indexing unlike strings, make sure that you are not changing your list in the loop in a way that causes you to index at or greater than the length of the list.

- For instance, if you decrease the length of the list and later index at a value that was valid before the change, it may be out of bounds now

My list changes unexpectedly

Make sure that you understand which list methods are destructive vs. non-destructive. Destructive methods will directly change your list (like pop() and append()) while non-destructive methods will return a value and leave the original list untouched (like choice()).

- Make sure you set a variable equal to the output of non-destructive methods; otherwise, there is no way to save the new list

Unit 11: 2D Lists and Board Games

Unit Review

2D Lists

Lists can contain anything—even other lists! This is invaluable for replicating tables, matrices, etc.

9	8	6
4	7	5

This table has 2 rows and 3 columns. We say that this is a 2x3 (read "2 by 3") table. Here is the same table with the rows and columns labeled:

	column 0	column 1	column 2
row 0	9	8	6
row 1	4	7	5

In Python, we can make this table using a **two-dimensional list**, often just called a **2D list**, like so:

```
L = [ [ 9, 8, 6 ],
      [ 4, 7, 5 ] ]
```

To access an element in a 2D list, use L[row][column] like below.

- L[0][0] == 9
- L[1][2] == 5

To quickly make a 2D list filled with all 0's, call makeList(rows, columns).

Remember nested for loops? They're great for looping through 2D lists.

```
for row in L:
  for element in row:
    genericFunction(element)
```

Vocabulary

2D Lists

2D Lists: Nested lists used to store a table of values.

- Ex.

0	1	2
3	4	5

  Represented in Python by [[0, 1, 2], [3, 4, 5]]

makeList(row, col): Returns a 2D list filled with 0's with the provided number of rows and columns.

Troubleshooting/Advice

I got an out of bounds error

With 2D lists, it can be very easy to confuse rows with columns, which can lead to invalid indexes. Double check your code to see if you accidentally switched these values.

Unit 12: Final Project

Unit Review

Images

To load an image, call Image(url, left, top). Store it as a local variable to change its width, height, and position properties, just as you would for a rectangle.

Sound

To load a sound, call Sound(url) (***The url must end in .mp3***). You can then call sound.play(), sound.pause(), and sound.stop() accordingly.

Vocabulary

Images

Image(url, x, y, width=a, height=b): Given a url, an x, and a y, returns and draws an image with its top left corner at the point (x, y). (You can also provide a specific width and height for the image to take, but this is optional.)

Sound

Sound(url): Takes the url of an mp3 file and then returns that sound, which can be stored in a variable.
sound.play(): Begins playing a sound either from the start, or if used after a pause, from where it left off.

- **sound.play(restart=True):** Always plays a sound from the beginning, even if it was paused previously.
- **sound.play(loop=True):** Plays the sound on a loop forever.

sound.pause(): Pauses a sound after it has started playing.

Troubleshooting/Advice

I'm using an image but it doesn't show up

This can be due to an invalid image url, in which case you need to find a new link or a new image.

It can also happen when the background of an image is see-through or the image color matches the background color. In this case, you will need to change the background color of the canvas or find a new image that is a different color.

My sound doesn't work

For our interface to process sounds, the website must end with .mp3. Any link with a different suffix won't work. If this is the problem, you will have to find a new sound or url.

Can I use my own image/sound?

We do not have a way for you to upload an image directly from your computer, but you can use an image that is uploaded to Google Drive. To do this, follow these steps:

1) Upload the image to Google Drive using your Gmail/GSuite account.
2) Get a sharing link by right-clicking the image and selecting the appropriate item from the menu. This link will look something like one of the following two links (depending on the sharing permissions):

   https://drive.google.com/open?id=1g-7WeWVs6OPBL7wqcZQx0z1neLvd_D9o
   https://drive.google.com/file/d/1g-7WeWVs6OPBL7wqcZQx0z1neLvd_D9o/view?usp=sharing

   Either way, the important part of the sharing link is the URL, which is the long string of random letters and numbers. In the example, that is '1g-7WeWVs6OPBL7wqcZQx0z1neLvd_D9o'.

3) Copy this string of letters and numbers onto the end of 'http://docs.google.com/uc?export=open&id='. In the example this would look like http://docs.google.com/uc?export=open&id=1g-7WeWVs6OPBL7wqcZQx0z1neLvd_D9o. This is the link that you can use to add the image to the canvas.
4) Here is some code that uses this example to add the image:

```
app.url =
'http://docs.google.com/uc?export=open&id=1g-7WeWVs6OPBL7wqcZQx0z1neLvd_D9o'
img = Image(app.url, 200, 200)
img.width = 400
img.height = 250
img.centerX = 200
img.centerY = 200
```