Carnegie Mellon University
Computer Science Academy

CMU CS Academy
AP CSP Teacher Handbook

Python Drop-in for Code.org's CSP Programming Units

Table of Contents

Course Description

Carnegie Mellon University has developed, in consultation with Code.org, an alternative option for Code.org's 21-22 AP CSP course for teachers who want to teach the programming units using CMU CS Academy's Python offerings. Teachers using this option will teach using Code.org's AP CSP materials for all of the units with the exception of the programming units (units 3, 4, 5 and 7), which use JavaScript. For these programming units, students and teachers will work from the CMU CS Academy platform and program in Python.

For teachers choosing this curriculum option, we highly recommend attending both the Code.org AP CSP and CMU CS Academy's AP CSP professional learning programs.

Information on Code.org's AP CSP course can be found on their website at https://code.org/educate/csp

For more information in general regarding the AP CSP course, please consider the course and exam description provided by the CollegeBoard, which can be found here: https://apcentral.collegeboard.org/pdf/ap-computer-science-principles-course-and-exam-description.pdf

The sequence consists of four units followed by the AP CSP Create Performance Task submission and Test Prep. We also provide Supplemental Materials. Each of the four units has a set of interactive notes, some of which are optional, and accompanying exercises, as well as a Practice Create Task. The optional notes and Supplementary Materials are not necessary to prepare students for the AP Exam.

The following table contains a high-level preview of the contents of each unit. The corresponding code.org unit number is also provided.

Code.org Number - CS Academy Unit Name	Topics
Unit 3—Intro to CMU Graphics	Drawing Basics, More About Drawing, and Create Task Design
Unit 4—Functions, Mouse Events,	Functions, Mouse Events, Properties, If-else

Conditionals	Conditionals, Custom Properties, If-elif-else, and Shape Methods
Unit 5—Groups, Lists, and Loops	Groups, Traversing Groups (Loops), Lists
Unit 7—Complex Conditionals, More Events, and Libraries	Complex Conditionals, Key Events, Step Events, Strings, Libraries, and Using Media
Unit 8—Create Performance Task *additional resource to use along with code.org Unit 8*	Create Task Examples, Practice, and Submission
Optional Unit—AP Test Prep	Test Prep and Exam Concept Review
Optional Unit—Supplemental Materials	Nested Loops, 2D Lists, and Project Practice

Commonly Asked Questions

Does using a Group count as using a List in the Create Performance Task?

Yes, it does! We do recommend students include a line at the start of their written response to 3b explaining why a Group is a “Collection type” like lists. The easiest way to do this is to remark that Groups are a list-like object with additional methods to assist with graphics.

Do event functions count as a “student-built” function?

No, students must write their own function, and the College Board explicitly states that Event Functions do not count.

How do students transition from the structured exercises in the course to building their own project for the Create Performance Task?

That’s what the Creative Tasks at the end of every unit are for! These are open-ended projects where the students can create their own project, allowing them to be used as practice Create Performance Task.

Are there any Examples of CS Academy Create Performance Projects?

We have created some examples, with written responses and grading comments, found in this document.

Have a question not on here?

Post it to the Teacher Forum or our Support Team, both found in your Teacher Portal’s Community Tab!

The Create Performance Task

Some Project Ideas

These projects are representative of the level of project we anticipate a student could create based on their completion of the curriculum.

Please note that not all of these projects would satisfy the Create Performance Task requirements, at least in their simplest form. In particular, the projects marked with a * below do not include lists and/or loops in the base version, although they could be added.

~ Level 1 Project	
Mad-libs*	Strings and app.getTextInput
Drawing Apps*	Conditionals and Variables
Asteroids	Groups, Conditionals, Loops, angleTo/getPointInDir functions
Flappy Bird	Groups, Conditionals, Loops
~ Level 2 Project	
Frogger	Groups, Conditionals, Loops
Wordle*	Strings, Key events, Conditionals
Escape Room*	Groups, Nested conditionals, Custom properties
~ Level 3 Project	
Fruit Ninja	Randomness, Groups, Lists, Loops, Motion
Whack a Mole	Randomness, Conditionals, Loops

Air Hockey*	Randomness, Motion, Mouse events, and Conditionals (no Loops)
Advanced (well-known games, but challenging and require 2D Lists)	
Tic-Tac-Toe	2D lists, Mouse events, Groups, and Helper functions
Simon	2D lists, Nested conditionals, and Helper functions
Word Search	2D lists, Nested for loops, Helper functions, and Key events
2048	2D lists, Mouse events, Conditionals, and Helper functions
Checkers	2D lists, Nested for loops, Groups, Conditionals, and Mouse events
Tetris	2D lists, Nested for loops, Helper functions, Groups, and Randomness

Material By Unit

Unit 1: Intro to CMU Graphics

Unit Review

Shapes You Can Draw

Shapes with radius

- **Circle(centerX, centerY, radius)**
- **Star(centerX, centerY, radius, points)**

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. Note that only fill, border, and borderWidth are introduced in Unit 1:

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)

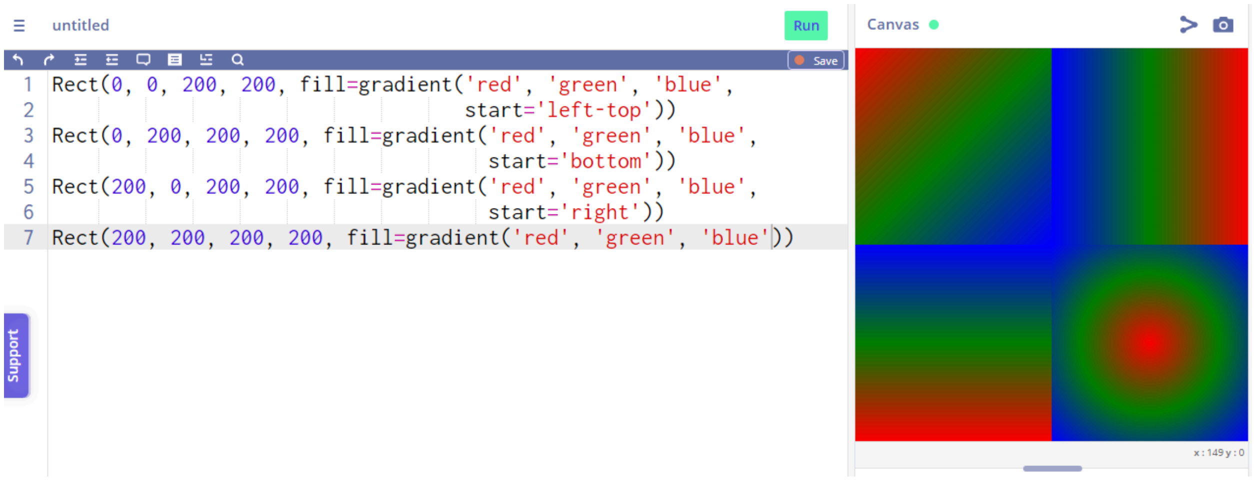
size—Controls the font size of the text in a label. 12 by default. (Only used on labels)

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).

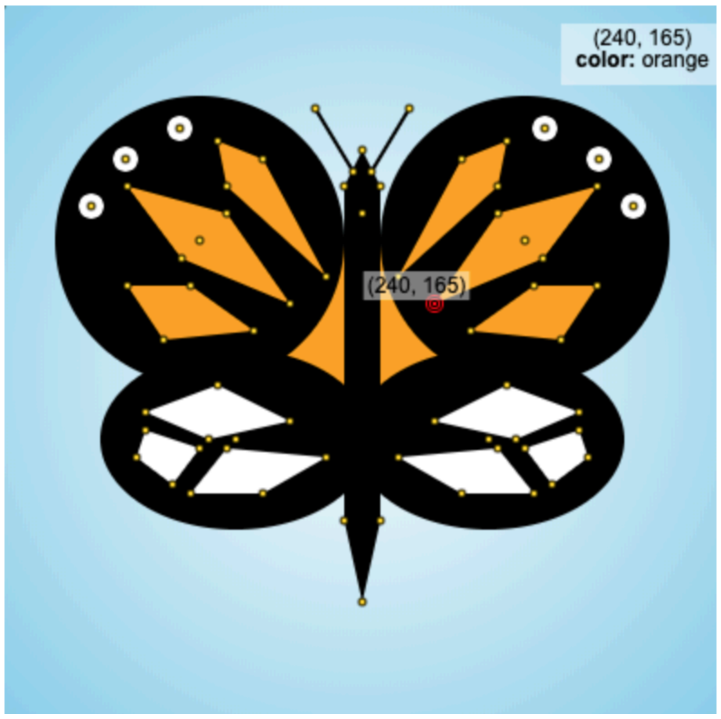


```
Rect(0, 0, 200, 200, fill=gradient('red', 'green', 'blue',
                                   start='left-top'))
Rect(0, 200, 200, 200, fill=gradient('red', 'green', 'blue',
                                     start='bottom'))
Rect(200, 0, 200, 200, fill=gradient('red', 'green', 'blue',
                                     start='right'))
Rect(200, 200, 200, 200, fill=gradient('red', 'green', 'blue'))
```

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

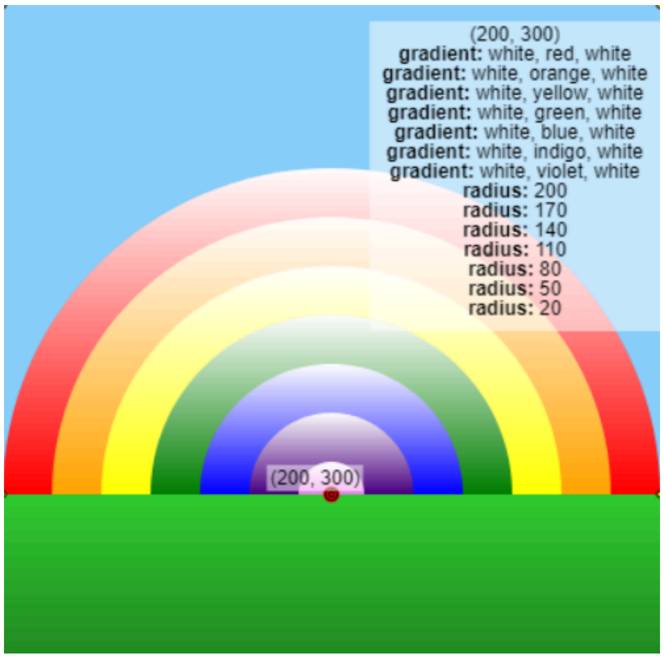


Once something has been drawn on the canvas, hover your mouse over some of the shapes. You will notice that some points are highlighted and that there is a grey box in the top-right corner with information in it, as shown in the image above. This is called the inspector and is vital to CMU CS Academy.

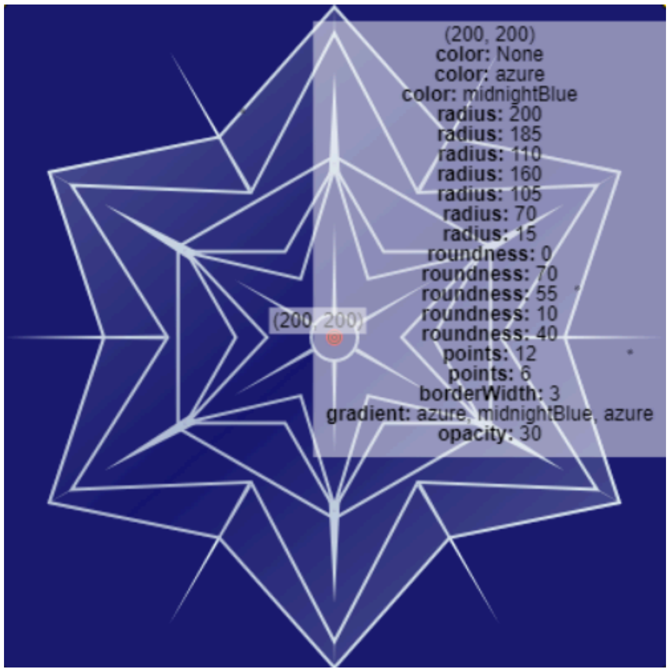
- Once we get further into the course and make interactive canvases, hold control to activate the inspector.

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.


(200, 300)
gradient: white, red, white
gradient: white, orange, white
gradient: white, yellow, white
gradient: white, green, white
gradient: white, blue, white
gradient: white, indigo, white
gradient: white, violet, white
radius: 200
radius: 170
radius: 140
radius: 110
radius: 80
radius: 50
radius: 20

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


(200, 200)
color: None
color: azure
color: midnightBlue
radius: 200
radius: 185
radius: 110
radius: 160
radius: 105
radius: 70
radius: 15
roundness: 0
roundness: 70
roundness: 55
roundness: 10
roundness: 40
points: 12
points: 6
borderWidth: 3
gradient: azure, midnightBlue, azure
opacity: 30

Important Tip: All properties are listed by the inspector from the back of the canvas to the front**.** In the rainbow example, the first radius shown is 200. So the circle that is drawn first (the red one) has a radius of 200.

Creative Tasks

The Creative Tasks at the end of every unit are intended to allow students to practice designing and creating their own programs from scratch. Even without many of the coding concepts needed for the Create Performance Task, creating their own open-ended programs is an important thing to practice, as it is a skill in and of itself.

We provide some additional documents to help students ease into the CSP Create Performance Task.

- Practice Creative Task Guide
- Create Performance Task Guide

AP Exam Vocabulary

- A program is a collection of program statements that performs a specific task when run by a computer. A program is often referred to as software.
- A code segment refers to a collection of program statements that are part of a program.
- Program behavior is how a program functions during execution and is often described by how a user interacts with it.
- Program output is any data sent from a program to a device. Program output can come in a variety of forms, such as tactile, audio, visual, or text.
- Program documentation is a written description of the function of a code segment, event, procedure, or program and how it was developed.
- Comments are program documentation for people to read that doesn't affect the code.
- An iterative development process requires refinement and revision based on feedback, testing, or reflection throughout the process. This may require revisiting earlier phases of the process.
- An incremental development process is one that breaks the problem into smaller pieces and makes sure each piece works before adding it to the whole.

Troubleshooting/Advice

Exercise drawing looks correct

Drawings must be *exact* matches. 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

Since grading is based solely on the canvas, make sure to use the inspector to its fullest extent. It's your best friend!

Note that most shapes can be drawn in multiple ways. For example, a rectangle may be drawn with either Rect or the Polygon function, and the autograder will accept both!

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 part of a comment '#'.

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

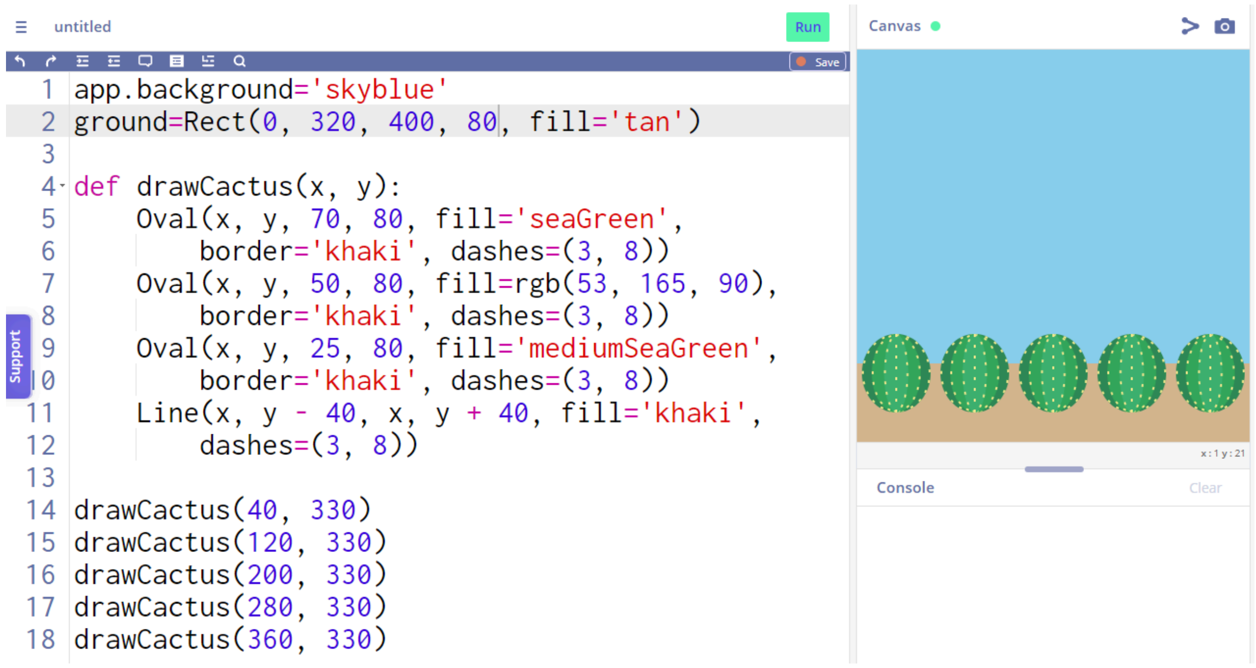
Layering is often used to cover up parts of shapes that aren't wanted. Code runs from top to bottom, so later lines of code get drawn on top of earlier ones.

Unit 2: Functions, Mouse Events, and Conditionals

Unit Review

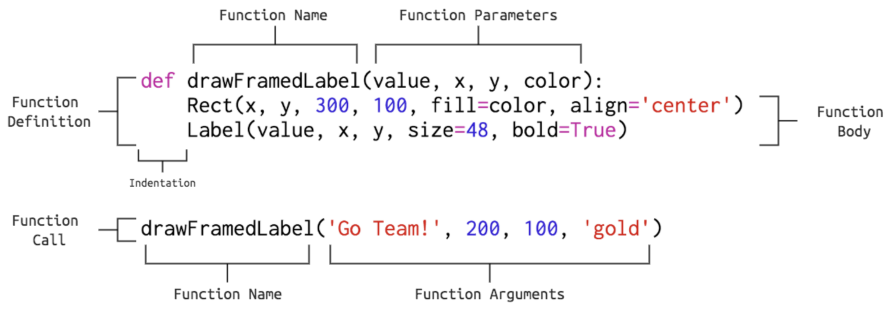
Functions

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:



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

def drawCactus(x, y):
    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=(3, 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))

drawCactus(40, 330)
drawCactus(120, 330)
drawCactus(200, 330)
drawCactus(280, 330)
drawCactus(360, 330)
```

Functions can have any number of parameters, that can be used to adjust how the function's body runs:



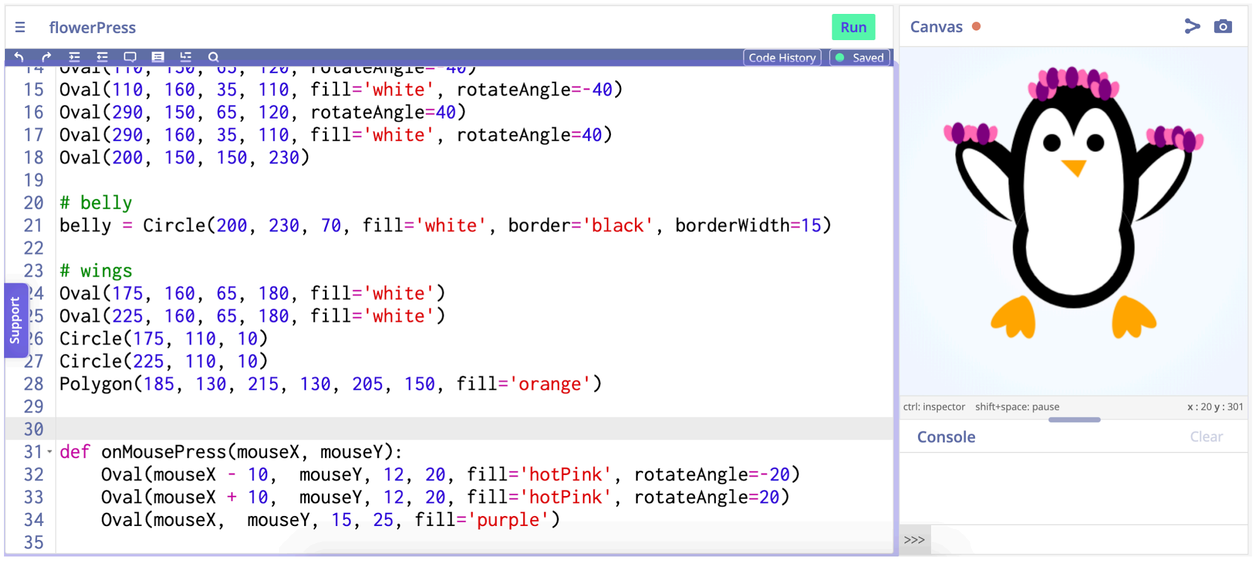
```
def drawFramedLabel(value, x, y, color):
    Rect(x, y, 300, 100, fill=color, align='center')
    Label(value, x, y, size=48, bold=True)

drawFramedLabel('Go Team!', 200, 100, 'gold')
```

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.

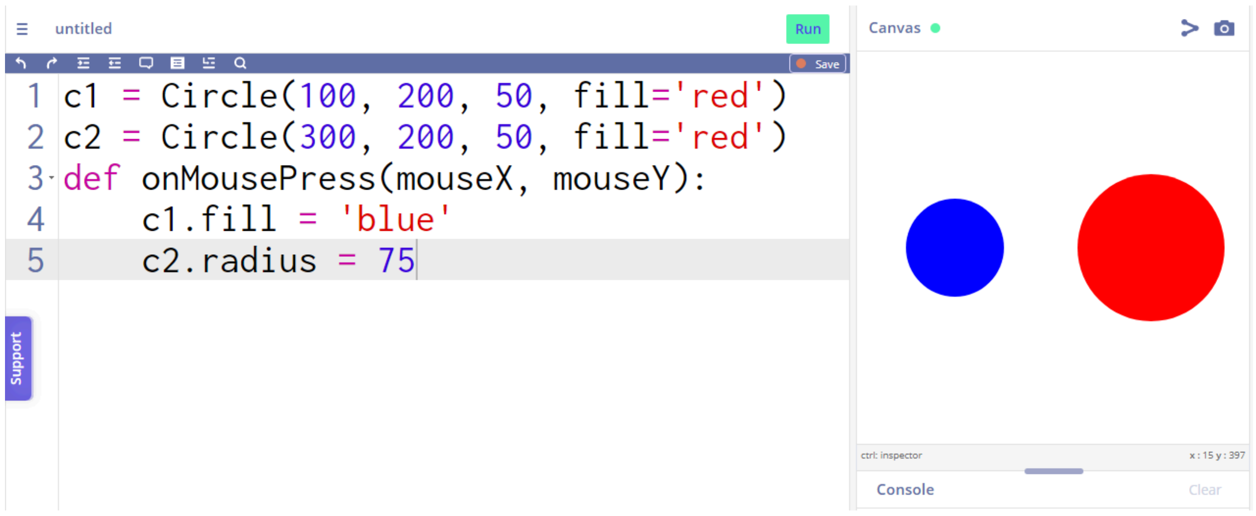
Using these event handlers, let's make a function that draws a flower wherever we press the mouse (see image below):



```
Oval(110, 150, 65, 120, rotateAngle=-40)
Oval(110, 160, 35, 110, fill='white', rotateAngle=-40)
Oval(290, 150, 65, 120, rotateAngle=40)
Oval(290, 160, 35, 110, fill='white', rotateAngle=40)
Oval(200, 150, 150, 230)

# belly
belly = Circle(200, 230, 70, fill='white', border='black', borderWidth=15)

# wings
Oval(175, 160, 65, 180, fill='white')
Oval(225, 160, 65, 180, fill='white')
Circle(175, 110, 10)
Circle(225, 110, 10)
Polygon(185, 130, 215, 130, 205, 150, fill='orange')


def onMousePress(mouseX, mouseY):
    Oval(mouseX - 10,  mouseY, 12, 20, fill='hotPink', rotateAngle=-20)
    Oval(mouseX + 10,  mouseY, 12, 20, fill='hotPink', rotateAngle=20)
    Oval(mouseX,  mouseY, 15, 25, fill='purple')

```

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.



```
c1 = Circle(100, 200, 50, fill='red')
c2 = Circle(300, 200, 50, fill='red')
def onMousePress(mouseX, mouseY):
    c1.fill = 'blue'
    c2.radius = 75
```

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

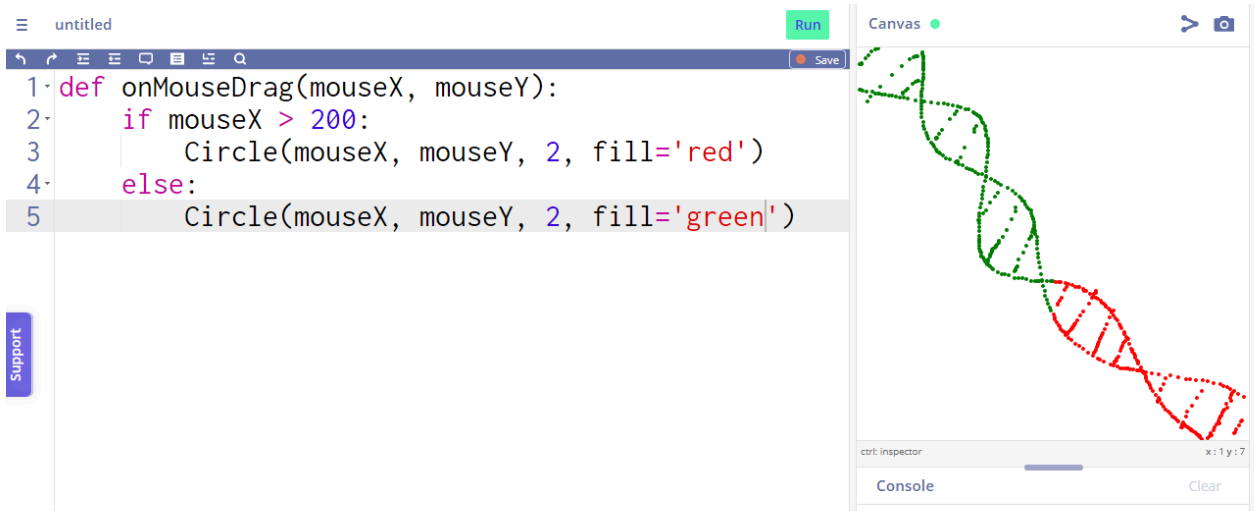
Size Properties: width, height, radius
Position Properties: left, right, top, bottom, align
Fill Properties: fill, opacity

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)

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:



```
def onMouseDrag(mouseX, mouseY):
    if mouseX > 200:
        Circle(mouseX, mouseY, 2, fill='red')
    else:
        Circle(mouseX, mouseY, 2, fill='green')
```

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.

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.

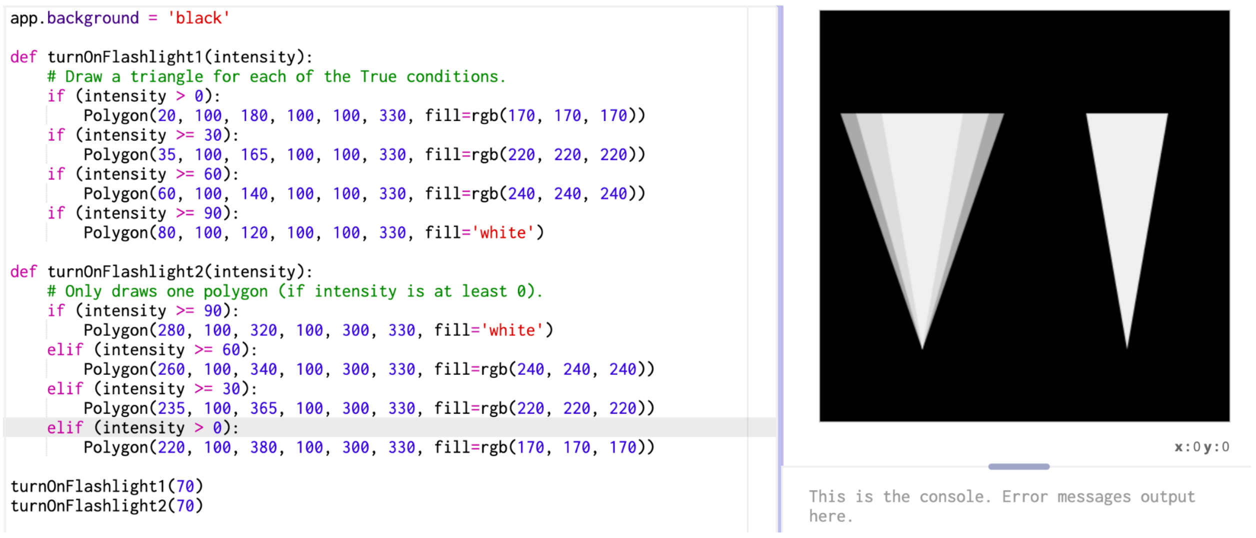
For multiple shapes:

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

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

It might be unclear when to use if-elif-else and when to use multiple if's, one after another. There is an easy way to determine this:

- If-elif-else should be used if you want the body of the first encountered True condition run.
- Multiple if statements should be used if you want the body of all True conditions to run.



```
app.background = 'black'

def turnOnFlashlight1(intensity):
    # Draw a triangle for each of the True conditions.
    if (intensity > 0):
        Polygon(20, 100, 180, 100, 100, 330, fill=rgb(170, 170, 170))
    if (intensity >= 30):
        Polygon(35, 100, 165, 100, 100, 330, fill=rgb(220, 220, 220))
    if (intensity >= 60):
        Polygon(60, 100, 140, 100, 100, 330, fill=rgb(240, 240, 240))
    if (intensity >= 90):
        Polygon(80, 100, 120, 100, 100, 330, fill='white')

def turnOnFlashlight2(intensity):
    # Only draws one polygon (if intensity is at least 0).
    if (intensity >= 90):
        Polygon(280, 100, 320, 100, 300, 330, fill='white')
    elif (intensity >= 60):
        Polygon(260, 100, 340, 100, 300, 330, fill=rgb(240, 240, 240))
    elif (intensity >= 30):
        Polygon(235, 100, 365, 100, 300, 330, fill=rgb(220, 220, 220))
    elif (intensity > 0):
        Polygon(220, 100, 380, 100, 300, 330, fill=rgb(170, 170, 170))

turnOnFlashlight1(70)
turnOnFlashlight2(70)
```

```
This is the console. Error messages output
here.
```

AP Exam Vocabulary

- A **procedure** is a named group of programming instructions that may have parameters and return values.
- **Parameters** are input variables of a procedure.
- **Arguments** specify the values of the parameters when a procedure is called.
- **Testing** uses defined inputs to ensure that an algorithm or program is producing the expected outcomes. Programmers use the results from testing to revise their algorithms or programs.
- A **code statement** is a part of program code that expresses an action to be carried out.
- An **expression** can consist of a value, a variable, an operator, or a procedure call that returns a value.
- **Program input** is data sent to a computer for processing by a program. Input can come in a variety of forms, such as tactile, audio, visual, or text.
- **Events** are associated with an action and supplies input data to a program.

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.

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 conditionals still don't work

It can be helpful to use print statements before your conditionals to check that your program values are what you think they are, or in the conditional bodies to see which condition is being used.

My code is hard to read

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

Custom properties aren't modified as intended

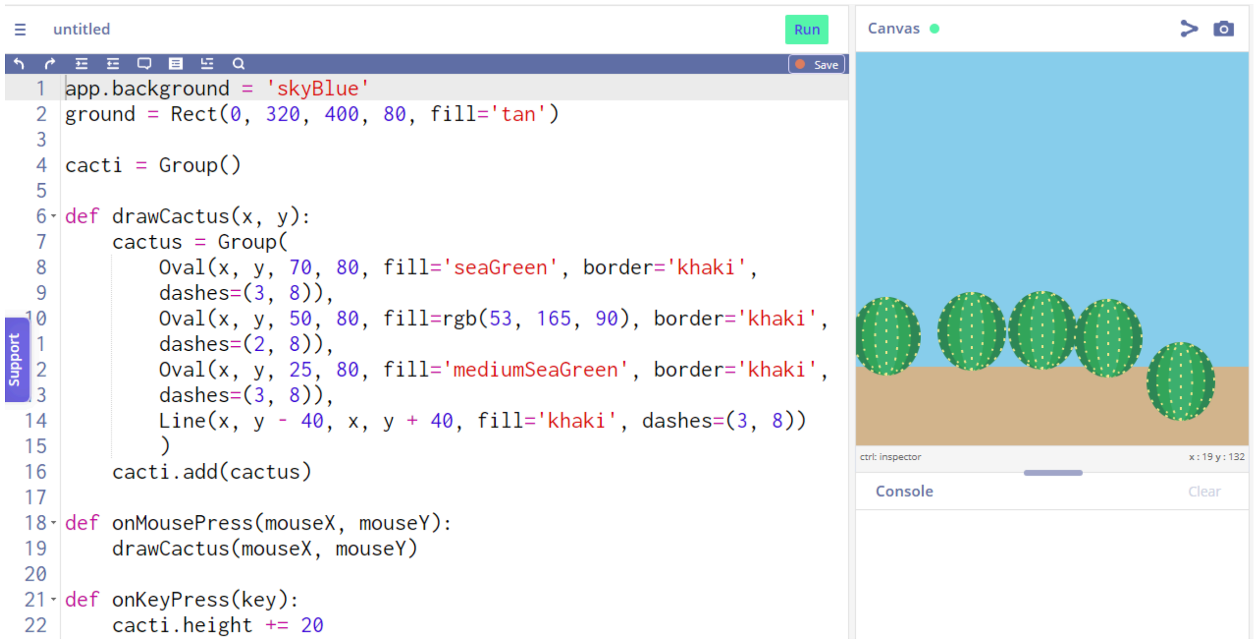
Make sure you have spelled your properties correctly; otherwise Python just makes the misspelling a new property.

Unit 3: Groups, Traversing Groups, and Lists

Unit Review

Groups

Groups are great for… well, grouping multiple shapes together. This lets us change those shapes' properties all at once. Here's an example where with a single key press we can increase the height of all cacti that we've drawn:



```
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
```

Groups have all of the general shape properties, except for border, borderWidth, and dashes. Similarly, all general shape methods can be used with groups, affecting <u>every</u> 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

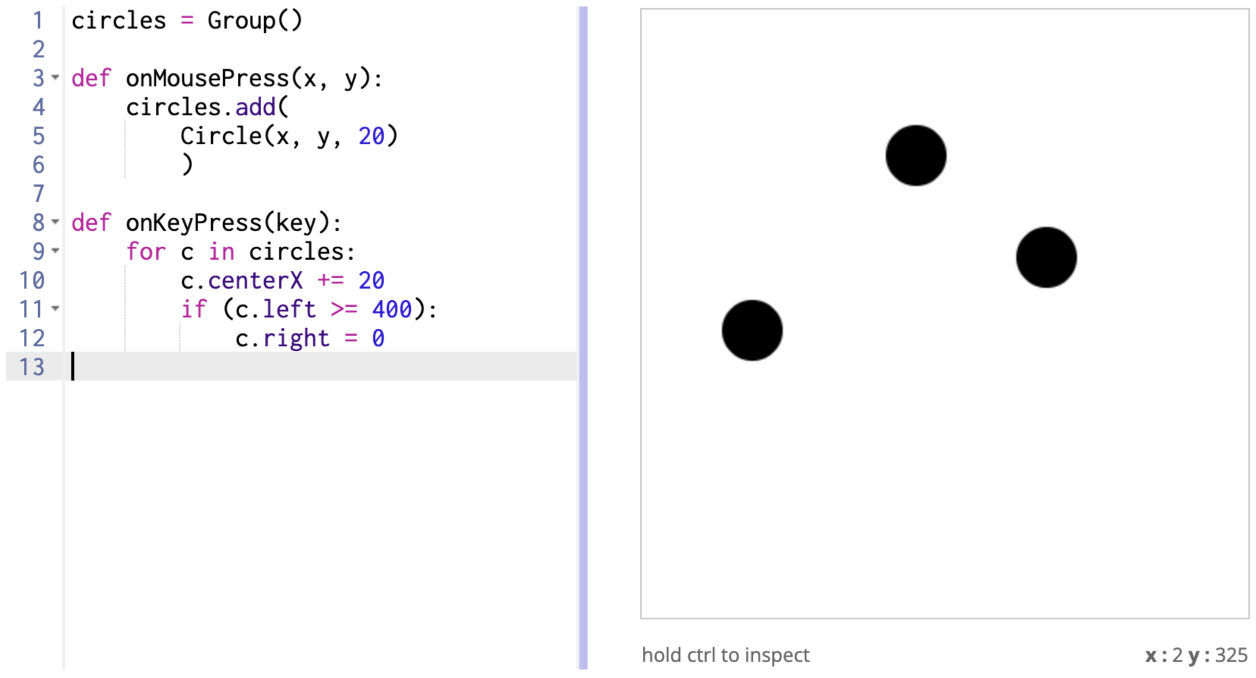
- **group.remove**(shape): If you give a shape a variable name, say s, you can later remove it from the group by calling group.remove(s).
    - This will also delete the shape from the canvas.
- **group.clear**(): Removes all shapes currently in a group
    - Just like remove, this deletes every shape from the canvas.

Traversing a Group

We can change every shape in a group 1-by-1 using a **for loop**.

```
for tempShapeName in groupName:
    # loop body here
```

Any code that is inside the loop body gets run once for each shape in the group, storing the shape in the `tempShapeName` variable. This way, we can change the properties of each shape in the group 1-by-1 by looping and changing the tempShapeName variable's properties.



```
circles = Group()

def onMousePress(x, y):
    circles.add(
        Circle(x, y, 20)
        )

def onKeyPress(key):
    for c in circles:
        c.centerX += 20
        if (c.left >= 400):
            c.right = 0

```

In this example, pressing any key will move every shape in the circlesGroup to the right. If any shape moves off the canvas (`c.left >= 400`), that shape then gets moved back to the left side of the canvas (`c.right = 0`).

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 = [ element1, element2, … ]
```

We can get an element at a particular position in a list using indexing. The syntax for this is

```
val = lst[index]
```

Important: In Python (and many other languages), the first element in a list is at index 0. However, on the AP CSP exam, the first element in a list is at index 1.

We can loop through a list just like we loop through a group:

```
for element in listName:
    # loop body
```

We can also loop through the possible indexes of a list using *range* and the `len` function.

```
for index in range(len(listName)):
    # loop body
```

The `len` function returns the number of elements in the list (len is short for length). When looping with range, the loop body runs as many times as is provided to the range. The combination of these two means that the loop runs once for every element in the list. But, instead of storing the element in the loop variable `index`, we store the numbers from 0 to `len(listName)-1` (ie, the valid indexes of `listName`).

List also have their own methods:

- **L.append(element):** Adds the element to the end of a list
- **L.insert(element, index):** Places the new element at the provided index in the list.

- **L.pop(index):** Removes *and returns* the element at the provided index. If no index is given, it takes the element from the very end of list
- **L.remove(element):** Removes the first occurrence of the element in the list

AP Exam Vocab

- A variable is an abstraction inside a program that can hold a value. Each variable has associated data storage that represents one value at a time, but that value can be a list or other collection that in turn contains multiple values.
- An index is a common method for referencing the elements in a list or string using natural numbers.
- An element is an individual value in a list that is assigned a unique index.

Troubleshooting/Advice

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. In addition to this, when a group contains labels, the width and height properties cannot be changed.

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.

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

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 your index never gets greater than or equal to the list length. Since lists *can* be changed by indexing, 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 4: Complex Conditionals, More Events, and Libraries

Unit Review

Complex Conditionals

Complex conditionals are conditionals that combine several, simpler tests. Compound conditionals contain the keywords and, or, or not. Nested conditionals are conditionals that contain other conditionals inside of their bodies.

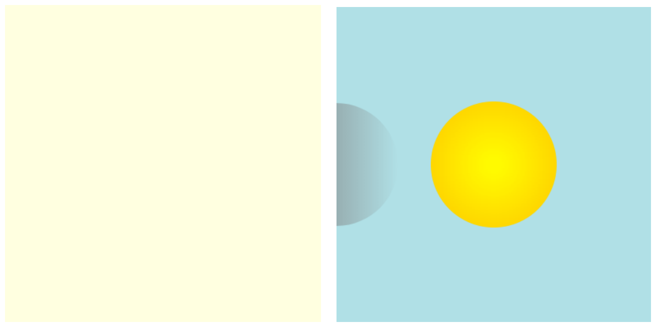
- **and:** Can be used to test if two conditionals are both true.
  - For example, `if ((isSummer == True) and (isSunny == True))` tests if it is both summer and sunny outside.
- **or:** Can be used to check if at least one of two conditionals is true.
  - For example, `if ((isSummer == True) or (isSunny == 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`) is `False` and (`not False`) is `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.

More Events

Just as you can use your mouse to interact with the canvas, you can also use your keys.



The 3 key event methods taught in this unit are `onKeyPress(key)`, `onKeyRelease(key)`, and `onKeyHold(keys)`. For the first two, the `key` parameter stores a string of what key was pressed or released. For onKeyHold, the `keys` parameter is a list of key strings. We can check if a key is being held, using the keyword "in."

```
    if (‘x’ in keys)
```

This unit also introduces onStep, which gets called 30 times every second by default. onStep is mostly useful for animations or games, because we can smoothly move or change shapes in onStep.

Strings

Strings are bits of text, contained within a pair of quotation marks (‘ ’ or “ ”). They are one of the basic types of data. Some other types are:

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`
lists	

We can combine two strings, called concatenating, by using the + sign.
`‘abc’ + ‘def’` becomes `‘abcdef’`.

We can loop over strings, just like with lists. Every iteration of the loop, the looping variable stores the next character in the string.

```
for c in s:
    if (c == ‘a’):
        ...
```

We can also index into a string, just like with lists. The ith index is the ith character in the string.
`s = ‘Hello!’`
`s[0]` is `‘H’`, `s[1]` is `‘e’`, etc.

Libraries

A library is a collection of pre-built functions and variables that can be used in our code so that we don’t have to build them ourselves. An example of this is the `random` library, which has functions like `randint` and `choice` that return random values.

To tell python we want to use a function from a library, we first need to import the library:
`import random`

After that, the library’s functions can be used with the following syntax:
`libraryName.libraryFunction()`

AP Exam Vocab

- A **string** is an ordered sequence of characters.
- A **substring** is a string that is a part of an existing string.

Troubleshooting/Advice

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 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 5: Create Performance Task

This unit outlines the Create Performance Task requirements and prompts, as well as provides some practice with creating larger projects.

Some additional useful resources can be found here:

- Create Performance Task Guide
- Sample Create Performance Tasks

The requirements of the Create Task project can be found either in the notes, or below:

- Functions (2 points) - Your project must use a function that:
    - is a function that you wrote. (not onMousePress or another event function!)
    - is called at least once in your code.
    - has at least one parameter.
    - changes what code it runs depending on the argument.
- Loops (1 point) - Your project must use a loop that:
    - includes an if Statement.
    - is inside a function that you have written.
    - Note that cycling through a list in onMousePress does **not** count as a loop!
- Lists/Groups (2 points) - Your project must use a list or group that:
    - stores data which is used in the project.
    - is important to the project. This means that writing the program without the list/group would have been much more difficult or impossible.

Appendix A: AP Test Prep

Unit Review

Abstraction

Abstraction is the process of removing details to create a more general representation. There are two types of abstraction that the AP Exam focuses on:

- Data Abstraction is when you remove details to represent values more generally.
- Procedural Abstraction is used to ignore the details of what your code is doing.

Algorithms

An algorithm is a detailed description of how to complete a task. It can be described either in english, or with **pseudocode**

On the AP Exam, they consider an algorithm to include the following three components:

- Sequencing is the order that the steps of an algorithm are given.
- Selection uses a boolean condition to determine which of two parts of an algorithm is used (Ex. conditionals).
- Iteration is when a part of an algorithm is repeated, either for a certain number of times or until a condition is met (Ex. for loops).

Robot

The AP CSP Exam includes a few questions that utilize the Robot. These questions typically involve a simple maze-like grid. We have implemented an animated version of Robot in our framework so students can practice reasoning about that type of setting.

Robot has a few basic methods of movement:

.

Instruction	Explanation
MOVE_FORWARD()	The robot moves one square forward in the direction it is facing.
ROTATE_LEFT()	The robot rotates in place 90 degrees counterclockwise (ie. makes an in-place left turn).
ROTATE_RIGHT()	The robot rotates in place 90 degrees clockwise (ie. makes an in-place right turn).
CAN_MOVE(direction)	

Most questions require students to either determine whether a provided bit of code will take the robot to the end goal, or identify the correct bit of code to take the robot to the end goal (from a couple options).

While Loops

A while loop is a type of loop that runs until a condition is met. For example, we can continue looping until a random value is exactly 50

```
import random

randNum = random.randint(0, 100)
while (randNum != 50):
    randNum = random.randint(0, 100)
```

This will keep checking if the new random number is 50, and if it is then the loop ends. But if it isn't, then we get a new random number and try again.

The main thing to look out for with while loops are situations where an infinite loop might occur. This is when the loop condition never is False, meaning the loop continues executing forever, or until the user ends it. The most basic type of infinite loop is:

```
num = 0
while (True):
    num += 1
```

This will continue adding 1 to the number forever.

AP Exam Review

Appendix A-Specific

- The subdivision of a computer program into separate subprograms is called **modularity**.
- A **simulation** is a representation that uses varying sets of values to reflect the changing state of a phenomenon.

All AP Exam Vocab

General Programming

- A **program** is a collection of program statements that performs a specific task when run by a computer. A program is often referred to as software.
- A **code segment** refers to a collection of program statements that are part of a program.
- A **code statement** is a part of program code that expresses an action to be carried out.
- An **expression** can consist of a value, a variable, an operator, or a procedure call that returns a value.
- **Program behavior** is how a program functions during execution and is often described by how a user interacts with it.
- **Program output** is any data sent from a program to a device. Program output can come in a variety of forms, such as tactile, audio, visual, or text.
- An **iterative** development process requires refinement and revision based on feedback, testing, or reflection throughout the process. This may require revisiting earlier phases of the process.
- An **incremental** development process is one that breaks the problem into smaller pieces and makes sure each piece works before adding it to the whole.
- A **simulation** is a representation that uses varying sets of values to reflect the changing state of a phenomenon.

Debugging

- A **syntax error** is a mistake in the program where the rules of the programming language are not followed.
- A **Runtime error** is a mistake in the program that occurs during the execution of a program. Programming languages define their own run-time errors.
- A **logical error** is a mistake in the algorithm or program that causes it to behave incorrectly or unexpectedly.
- **Testing** uses defined inputs to ensure that an algorithm or program is producing the expected outcomes. Programmers use the results from testing to revise their algorithms or programs.
- **Comments** are a form of program documentation written into the program to be read by people and do not affect how a program runs.
- **Program documentation** is a written description of the function of a code segment, event, procedure, or program and how it was developed.

Procedural Abstraction

- A **procedure** is a named group of programming instructions that may have parameters and return values.
- **Parameters** are input variables of a procedure.
- **Arguments** specify the values of the parameters when a procedure is called.
- **Program input** is data sent to a computer for processing by a program. Input can come in a variety of forms, such as tactile, audio, visual, or text.
- **Events** are associated with an action and supplies input data to a program.
- The subdivision of a computer program into separate subprograms is called **modularity**.

Data Abstraction

- A **variable** is an abstraction inside a program that can hold a value. Each variable has associated data storage that represents one value at a time, but that value can be a list or other collection that in turn contains multiple values.
- An **element** is an individual value in a list that is assigned a unique index.

- An **index** is a common method for referencing the elements in a list or string using natural numbers.

Appendix B: Supplemental Materials

Unit Review

The materials in this unit are not relevant to the Create Performance Task or the AP CSP exam, but some students might find them useful for their projects.

Nested Loops

We've seen that we can use loops to run code multiple times. We can run an entire loop multiple times by putting inside another loop, like so:

```
for i in range(5):
    for j in range(3):
        print(i, j)
```

This is called nesting loops. The first loop (with i) is called the *outer* loop, and the second loop (with j) is called the *inner* loop. Every iteration of the outer loop, the inner loop fully runs. So in this example, `j` is 0 five different times (once each when `i` is 0, 1, 2, 3, 4).

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).

We can loop through a 2D list using a nested loop. The outer loop iterates over the 2D list's sublists, then the inner loop iterates over the elements in each sublist.

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

Additional Resources

AP CSP Test Prep

We recommend that you use the Official Sample Exam Questions that the CollegeBoard provides. You can find these questions in the CollegeBoard's AP Computer Science Principles Course and Exam Description document.

Sample Exam Questions