



Curriculum and Instruction

CTE COURSE OF STUDY OUTLINE

Title of Course of Study Computer Science 1 CTE

Course Number: _____ (Assigned by Curriculum Department)

| CTE Course of Study Adoption Process | |
|--------------------------------------|---|
| PROCEDURES: | |
| 1 | Write/revise course of study |
| 2 | Review with CTE Principal and acquire signature |
| 3 | Email course of study to appropriate CTE sector staff at all high schools with link to Curriculum Council survey |
| 4 | Attach copy of survey and comments along with sign in sheet from required department meeting. |
| 5 | Meet with appropriate department teachers to discuss responses, review course of study and sign. Attendance sheet of meeting is required |
| 6 | Technology review/sign prior to submission required if any technology components used |
| 7 | Course of study MUST be complete, including required signatures, and submitted to Curriculum Dept. 2 weeks prior to the scheduled Curriculum Council meeting. |
| 8 | Assistant Superintendent, Curriculum & Instruction - Review/Sign |
| 9 | Assistant Superintendent, Secondary Education - Review/Sign |
| 10 | Present course of study to Curriculum Council |
| 11 | Curriculum Council Recommends |
| 12 | Board of Education Approves |

Note: Please complete all sections. Enter "none" or "n/a" as appropriate.

I. Course Title: Computer Science 1 CTE

II. Industry Sector: Information and Communication Technologies

Pathway Name: System Programming Pathway No. 174A Calpads: 8131

- Introduction Course
- Concentrator Course
- Capstone Course

Course of Study Proposal Reason:

- New Course**
- Curriculum Update
- Textbook Update
- UC/CSU a-g Update
- Course Title Change

III. Length of Course: _____ Credit Value: _____

X Meets Lodi USD high school graduation requirement credits
 _____ Elective Course credit _____ No Credit

IV. Grade:

- 9th
- 10th
- 11th
- 12th

V. Course Level: General CP Honors Pre-AP AP

VI. Is this an Internet-based course? Yes No (e.g. Apex, Odysseyware...)
If so, who is the course provider?

VII. UC/CSU Approved Course:

Do you wish to submit this course to the UCOP to obtain UC/CSU a-g approval?

Yes No

Is this course modeled after a UC-approved course from another district?

Yes No If so, which school/district?

Da Vinci Charter Academy / Davis JUSD - Intro to Coding & CS

VIII. Recommended UC/CSU Subject Area Pathway:

(Please complete each section as required by the UC system)

- | | |
|---|--|
| <i>A. History/Social Science</i> | <i>E. Languages Other than English</i> |
| <i>B. English</i> | <i>F. Visual/Performing Art</i> |
| <i>C. Math</i> | <i>G. Elective</i> |
| <input checked="" type="checkbox"/> <i>D. Lab Science</i> | |

IX. Subject Area Code for Lodi USD Graduation Requirements (select all that apply):

- | | |
|--|------------------------------|
| <i>B. Fam Lvg/World Geography</i> | <i>L. Life Science</i> |
| <i>C. Economics</i> | <i>M. Mathematics</i> |
| <i>D. Driver's Ed</i> | <i>P. Physical Education</i> |
| <i>E. English</i> | <i>S. Physical Science</i> |
| <input checked="" type="checkbox"/> <i>F. Fine Arts/For Lang/CTE</i> | <i>U. US History</i> |
| <i>G. Government</i> | <i>W. World History</i> |
| <i>H. Health/Safety</i> | <i>Y. Elective</i> |

- X. COURSE DESCRIPTION:** *Use this section to emphasize the core knowledge and skills students are expected to learn in the course, including concepts, theory and texts. There should be clear evidence of the course's level of rigor and the development of essential critical thinking skills.*

The Course Description is comprised of three sections:

- 1. COURSE OVERVIEW:** *Students will learn to program classic video games (pong, pac-man, tetris, etc) utilizing Carnegie Mellon University's CS Academy curriculum (academy.cs.cmu.edu) and will conduct a series of hands-on experiments with micro:bit microcontrollers (microbit.org) emphasizing relevant Internet of Things (IoT) applications. Students will learn the basic constructs of programming, including variables, constants, expressions, control structures, functions and arrays. No prior programming knowledge or experience required.*
- 2. HIGHLY RECOMMENDED Co-REQUISITES:**
Integrated Math 1 or equivalent
- 3. COURSE CONTENT:**

Students will earn a "Certificate of Completion" from Carnegie Mellon University for completing the following 12 Units:

Unit 1: Create Drawings

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).

3 Weeks (16 Days)

- 1.1 Notes and Exercises (3 Days)
- 1.2 Notes and Exercises (3 Days)
- 1.3 Notes and Exercises (3 Days)
- 1.4 End of Unit Exercises (2 Days)
- Creative Tasks (3 Days)
- Review/Quizzes (2 Days)

Example Exercise Lighthouse

Unit 2: Functions, Mouse Events, and Properties

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.

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.

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.

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

2 Weeks (11 Days)

- 2.1 Notes and Exercises (2 Days)
- 2.2 Notes and Exercises (2 Days)
- 2.3 Notes and Exercises (1 Day)
- 2.4 End of Unit Exercises (1 Day)
- Creative Tasks (3 Days)
- Review/Quizzes (2 Days)

Example Exercise Eat the Cooke

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

Mouse Motion Events

In addition to `onMousePress` and `onMouseRelease`, we have `onMouseMove` and `onMouseDown`. 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 `onMouseDown(...)`.

Conditionals

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

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.

3 Weeks (13 Days)

- 3.1 Notes and Exercises (2 Days)
- 3.2 Notes and Exercises (2 Days)
- 3.3 Notes and Exercises (3 Days)
- 3.4 End of Unit Exercises (1 Day)
- Creative Tasks (3 Days)
- Review/Quizzes (2 Days)

Unit 4: More Conditionals, Key Events, and Methods

If-elif-else, vs. multiple ifs

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.

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()`: Adds a point to a regular polygon (i.e. pentagon -> hexagon).
- `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.

2 Weeks (12 Days)

- 4.1 Notes and Exercises (2 Days)
- 4.2 Notes and Exercises (2 Days)
- 4.3 Notes and Exercises (2 Days)
- 4.4 End of Unit Exercises (1 Day)

- Creative Tasks (3 Days)
- Review/Quizzes (2 Days)

Example Exercise Trippy Text

Unit 5: Complex Conditionals and More Key Events

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)`.

2 weeks (10 Days)

- 5.1 Notes and Exercises (2 Days)
- 5.2 Notes and Exercises (2 Days)
- 5.3 End of Unit Exercises (1 Day)
- Creative Tasks (3 Days)
- Review/Quizzes (2 Days)

Example Exercise Biking

Unit 6: Groups, Step Events, and Motion

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.

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.

3 weeks (15 Days)

- 6.1 Notes and Exercises (2 Days)
- 6.2 Notes and Exercises (2 Days)
- 6.3 Notes and Exercises (3 Days)
- 6.4 End of Unit Exercises (3 Days)
- Mid-Year/ Semester Creative Tasks (3 Days)
- Review/Quizzes (2 Days)

Example Exercise Hungry Giraffe

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

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).

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:

2 Weeks (12 Days)

- 7.1 Notes and Exercises (2 Days)
- 7.2 Notes and Exercises (1 Day)
- 7.3 Notes and Exercises (2 Days)
- 7.4 End of Unit Exercises (2 Days)
- Creative Tasks (3 Days)
- Review/Quizzes (2 Days)

Example Exercise Space Invaders

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

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.

3 Weeks (13 Days)

- 8.1 Notes and Exercises (2 Days)
- 8.2 Notes and Exercises (3 Days)
- 8.3 Notes and Exercises (2 Days)
- 8.4 End of Unit Exercises (1 Day)
- Creative Tasks (3 Days)
- Review/Quizzes (2 Days)

Example Exercise Compass

Unit 9: Types, Strings, and While Loops

Types

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

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.)

2 Weeks (12 Days)

- 9.1 Notes and Exercises (2 Days)
- 9.2 Notes and Exercises (2 Days)
- 9.3 Notes and Exercises (1 Days)
- 9.4 Notes and Exercises (1 Days)
- 9.5 End of Unit Exercises (1 Day)
- Creative Tasks (3 Days)
- Review/Quizzes (2 Days)

Example Exercise Sheep wool

Unit 10: Lists and Return Values

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)
```

2 Weeks (10 Days)

- 10.1 Notes and Exercises (1 Day)
- 10.2 Notes and Exercises (1 Day)
- 10.3 Notes and Exercises (2 Days)
- 10.4 End of Unit Exercises (1 Day)
- Creative Tasks (3 Days)
- Review/Quizzes (2 Days)

Example Exercise Whack a Bug

Unit 11: 2D Lists and Board Games

2D Lists

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

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 column in L:

 genericFunction(L[row][col])

2 Weeks (9 Days)

- 11.1 Notes and Exercises (2 Days)
- 11.2 Notes and Exercises (1 Day)
- 11.3 End of Unit Exercises (1 Day)
- Creative Tasks (3 Days)
- Review/Quizzes (2 Days)

Example Exercise Essay Editor

Unit 12: Final Project

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.

2-4 Weeks



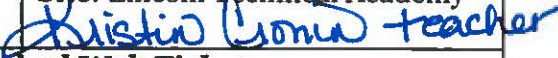

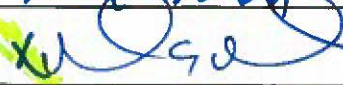






- 12.1 Notes (1 Day)
- End-of-Course Creative Task (10-20 Days)


XI. Texts and Supplemental Instructional materials:

(Primary, Supplemental, newspapers, magazines, and software.)

Please supply ISBN #'s for all texts.

Websites: <https://academy.cs.cmu.edu/>
<https://microbit.org/>
<https://learn.sparkfun.com/>

| SIGNATURES for REVIEW | | |
|---|---|---|
| Outline prepared by |  | Site: Lodi HS & Tokay HS |
| CTE Principal |  | Site: Lincoln Technical Academy  teacher |
| Technology Representative (if applicable) | See attached Web Ticket | |
| Teacher Representative: | <i>Signature indicates course is aligned to CTE Model Standards.</i> | ** Please state reason for no signature in the space below. |
| Bear Creek High School | Not Applicable | Not taught at this site & no current CS Teacher |
| Lodi High School |  | |
| McNair High School |  | |
| Tokay High School |  | |
| Principal | | |
| Lodi High School |  | |
| McNair High School |  | |
| Tokay High School |  | |
| Assistant Superintendent Curriculum & Instruction |  | |
| Assistant Superintendent, Secondary Education |  | |

| DATE | |
|---|--|
| 9/3/2021 | Date sent and/or presented to principal for review |
| 9/3/2021 | Course Outline Submitted |
|  | Curriculum Council Recommendation for Approval |
| | Board of Education Approval |