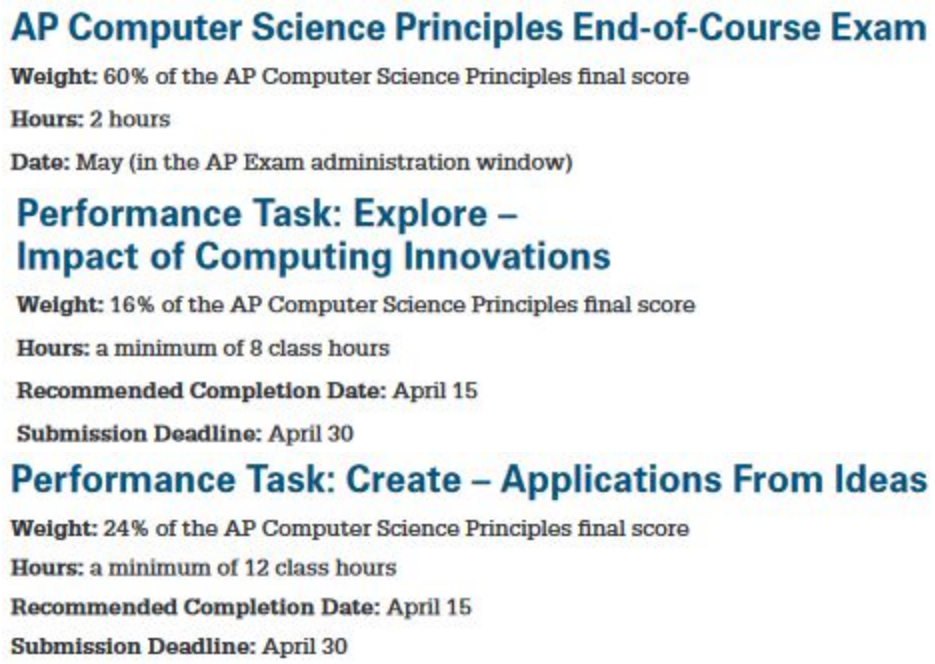
AP Computer Science: Principles

Python Curriculum/Overall Course Review

OVERVIEW

As you guys may already be aware of, the APCS:P score is broken up into three parts:



AP Computer Science Principles End-of-Course Exam

Weight: 60% of the AP Computer Science Principles final score

Hours: 2 hours

Date: May (in the AP Exam administration window)

Performance Task: Explore – Impact of Computing Innovations

Weight: 16% of the AP Computer Science Principles final score

Hours: a minimum of 8 class hours

Recommended Completion Date: April 15

Submission Deadline: April 30

Performance Task: Create – Applications From Ideas

Weight: 24% of the AP Computer Science Principles final score

Hours: a minimum of 12 class hours

Recommended Completion Date: April 15

Submission Deadline: April 30

The AP Exam for CS: P is on Friday, May 10th (1st week) in the afternoon.

Edhesive does a pretty good job of preparing for the exam itself. Sample questions can be found starting on page 85 of the AP Course Description. As such, the primarily focus will be on helping develop ideas for the Explore task, and the programming skills needed to tackle the Create task. We're going to leave our academic emails here; don't be afraid to shoot us an email asking for programming help or to review your Explore project (within reason):

rfaria@andrew.cmu.edu

macea@andrew.cmu.edu

Explore Task

What do I have to do?

Page 74 of the course description contains the formal explanation of the task assignment. As per the rubric on page 108, the short version is as such:

- The first part is the "computational artifact" itself, which can be a visualization, a graphic, a video, a program, or an audio recording.
- The second part is a report based on the invention you chose, and fulfills all the requirements on page 110. Since the maximum word count is 700, this shouldn't take longer than a page and a half.

How do I pick a technological invention?

Anything you can think of is probably okay. The only thing to keep in mind is that you have to be able to discuss one pro, one con, and how the invention manipulates data, and "at least one data storage concern, data privacy concern, or data security". With that being said, here's a list of links with ideas to get you started:

- https://compsci.lafayette.edu/top-30-innovations/
- https://www.fool.com/investing/2017/08/04/5-technological-innovations-that-could-change-the.aspx (#3 isn't recommended)

Example Projects

At the bottom of this page you'll find sample responses and scoring guidelines from CollegeBoard. Here is a sample project, with a sample of the submission format for the digital portfolio and the highlighting required for the code submission.

Create Task

For the create task, this is where you get to truly show off your creative skills if what you enjoy about computer science is the nitty gritty coding of it all. To get ready for this portion of the exam, we'll introduce you to the basic programming skills needed to create a project for your upcoming AP Portfolio.

Before we start, it's important to note that we're going to be using material from our Intro to Programming course at Carnegie Mellon, but we'll be showing you the extremely important parts that you need to know. In any case, when you're done here, feel free to go on the course's

website (linked below) in order to continue learning all the wonderful things there are to programming!

Fall 2018 Schedule: http://www.krivers.net/15112-f18/schedule.html

Getting Started with Python

For your create task we'll be working with the programming language called Python, for this language, there are a couple things you need to download before getting started.

First is the language itself, from the python website, download the latest version of Python.

Python Website: https://www.python.org/downloads/

Then, you'll need a text editor, or somewhere to write your code, for your create task we'll be using a software called Pyzo. We've linked the download folders below.

Pyzo for Windows

Pyzo for Mac OS

Pyzo for Linux

Setup your shell configuration for Python 3

- On the top menu, click Shell->Edit Shell Configurations
- Click the dropdown labeled exe, and choose Python 3.6.
- Under "gui", select "None no gui support" (otherwise you may have problems when we do graphics).
- Press done.
- Restart Pyzo and you're ready to go!

Close the extra frames (file browser, etc), only use editor frame (on top, where it is by default) and shell frame (which you should drag to the bottom), so it mirrors IDLE's simple setup.

Relaunch Pyzo, for a simple Python3 test, use this program: print(5/3), then choose Run File as Script or Execute File from the Run menu. You should not get 1 (which you would get if you are using Python2) but instead should get 1.666666667.

Running your code:

Each IDE has its own way to run code. In Pyzo, from the "Run" menu, select "Run file as script". This loads and runs your code in the Python Shell (the interpreter that runs Python code, instead of editing it).

- There are several options on the Run menu. Be sure to use the first one, "Run file as script".

Actually learning how to code!

Now that you can write, save, and run code. Feel free to open a file and practice along with the things we teach you.

Lesson 1 - Hello World

As a sort of stepping stone into what is code, every new coder gets the magical two words “Hello World” to show up on their screen. To do this python has a simple command called print() that prints something onto your screen when you run it.

Try writing and running this piece of code (don’t just copy it) :

```
print(“Hello World”)
```

What happens when you run it? What happens when you change the message in the quotation marks?

Throughout these lessons, there are going to be certain things that you have to trust from us in terms of just writing some code down first so you can later understand how it works, keep this in mind as you continue reading these. This lesson was meant to be an introduction to this sort of thing.

Lesson 2 (Really 1) - Basic Data Types

In Python, there are different ways to represent data, those include:

- Integers (1,2,3,etc. whole numbers)
- Floats (1.23445, 3.14, etc. decimals)
- Strings (Phrases) (Ex. “Marco is awesome”, “Hello World”)
- Booleans (True or False)

Later on, all programming does is manipulate these basic types of data and information in clever ways.

Examples - State which of the data types correspond to the following:

Answers are in white next to the questions, highlight to reveal

123 “I hope my code works” True

Bonus Problem: What data type is 143.0?

Lesson 3 - Math Operations in Python

Much like a calculator, you can use math to compute values and edit the numbers your program will play with in the future.

Python has all the basic Operators like:

- Minus (-)
- Plus (+)
- Divide (Just done with a “/”, this will return decimals if it has to)
- Multiply (Done with an asterisk i.e *)
- Exponents (In Python its written with two asterisks i.e **)

The more complicated operators are:

- Modulo, written with the % sign
    - Gives back the remainder of a division, so 3 % 2 equals 1
- Integer Division, written with two slashes, //
    - Doesn't give back decimals, so 3 / 2 which gives back 1.5, if done like 3 // 2 would just give back 1
    - It simply removes the decimal, it **DOES NOT ROUND**

Lesson 3.5 Mini Lesson - Variables

In python, the way you assign a variable is with the = sign, variables can hold any data type but for now, we’ll just play with numbers.

Try writing and running this piece of code (don’t just copy it) :

```
x = 36

print(x)
```

To change variables, just call that variable and set it equal to something else.

Try writing and running this piece of code (don’t just copy it) :

```
x = 36

print(x)
```

```
x = 38

print(x)
```

To alter a variable that already exists, like x = x + 10, there are shortcuts for this in python:

- x += something is like saying x = x + something
- x -= something is like saying x = x - something
- x /= something is like... you get the idea... but if you don't try writing your own code to figure it out!
- x *= multiplies something
- x //= integer divides something
- x %= modulus something

Lesson 4 - Comparators in Python

Much like in math, there are ways to compare numbers in math.

Here are all the comparators:

- < , Less than
- > , Greater Thank
- <= , Less than or equal to
- >= , Greater than or equal to
- == , is equal to
    - Why can't we use just one equal sign?
- != , not equal to
    - This is to check if things are **not** equal to each other
    - The word **not** can also be used return the opposite of a boolean
        - Try printing not True and not False to see what you get back!

Lesson 5 - Conditional Statements

In Python, there are ways to determine whether something happens if two or more things are true. Python uses special words like **and** & **or** to handle True and False statements.

For the **and** keyword, both arguments need to be True for it to return True, anything else returns False.

Examples (Test out your own cases!):

```
print( True and True)
```

```
print( 4 > 4 and 5 ==5)

      print(4 < 5 and 3 < 5)

print(2 != 2 and False)
```

For the or keyword, only one of the arguments need to be True.

Examples (I'll only give you one, try your own!):

```
print(True or False)
```

Mini-Lesson 5.5 - Commenting

It's important to leave comments on your code that the computer won't read so that you as a human can know what's going on in your code. This is done with the **#** symbol, which you have to do for every line that you comment on.

Example:

```
x = 3

#I wanted to write two lines of comments

#I want to print if x is equal to 3

print( x == 3 )
```

Lesson 6 - If, Elif, and Else Statements

In any program, there are certain conditions that determine its outcome, that is managed by if, elif, and else statements.

Basically, the way it works is that **if *some condition is met: it does something else***

Here is an example:

```
x = 3

if x == 3:

 print( "HI HOW ARE YOU?" )
```

*It is important that you indent after your if statement to show that its what happens if the condition is met

`else` is everything that happens if the if statement didn't go through.

Example:

```
x = 3

if x ==3 :

      print( “Nope” )

else:

      print( “X was not equal to 3” )
```

Elif is required to be sandwiched between if and else, because it works exactly like the if statement, but is just another case that you might be checking for.

Here is an example:

```
x = 3

if x == 4:

      print(“This won't run.”)

elif x == 3:

      print(“This will run”)

else:

      print(“It runs on the elif statement because x = 3”)
```

Lesson 7 - Functions

You’ve now learned all the basics needed to write your own functions! In our first lesson, print() is an example of a function, that takes in some data, and then outputs some other type of data.

To make a function, the syntax is:

```
def functionName(arguments):

#Here goes your code!
```

The print function probably looks something like this:

```
def print( data ):
    #Prints the data you gave it
```

As an example, let's write a function that determines whether a integer is odd or even:

```
def isOddOrEven( number ):
if number % 2 == 0:
        print( "This number is even!" )
else:
        print( "This number is odd!")
isOddOrEven( 3 )
```

Instead of printing something, we can actually make a function return some type of data by using the reserved word **return**.

For Example:

```
def isOddOrEven( number ):
    if number % 2 == 0:
        return "This number is even!"
    else:
    return "This number is odd!"
print( isOddOrEven( 3 ) )
```

Both examples have their advantages, for the first example, the program prints when you call it on its own, and for the second example you have to print the result of the function.

Now that you've learned about functions, you should try going on the Carnegie Mellon Website we linked above and try reading through the week 1 practice our professor wrote for us.

Lesson 8 - Loops

You're now reaching the peak of the learning curve in terms of how complicated things will get with the code you'll need for your projects.

There are two kinds of loops, **for loops** and **while loops.**

For loops basically, repeat code for the specific amount of times. The range(start, end) function creates a range of numbers from the start to the end-1.

To write a for loop the syntax goes (**pseudocode):**

```
for variable in range( start , end ):

#Write the code that will be repeated
```

Try writing and running this code (What happens if you only write one number in range?):

```
for x in range(0,5):

      print ( x )

#The range function also has a step parameter

#This steps over the range by 2 instead of 1

for x in range(0,10,2):

      print(x)
```

While loops repeat a certain piece of code until a condition is met

To write a while loop the syntax goes (**pseudocode)**:

```
while( condition ):

#Write the code that will be repeated
```

Try writing and running this code (What happens if the condition is always met? What if it is never met, does it run at least once?):

```
x = 3

while (x > 0 ):

print( "YES!" )
```

```
            x -= 1

        #What if you take out the x -= 1 ? Or if you set x equal to -1 at the
beginning?
```

Mini-Lesson 8.5 - Testing your code

How can you make sure your function is working okay? How do college classes check your work? The answer is, **assert** statements.

Whenever you have a problem in your code, or it's not working as you expect it to, that is called a **bug**.

You could print the outcome of your function and check what if it's what you expected, OR you could use an **assert statement** which compares the result of a call to a function with your expected result.

Try writing and running this code:

```
    def practice(score):
        if score >= 90:
          return "A"
        elif 80 <= score < 90:
          return "B"
        else:
          return "What else should we check?"
    #This function takes now parameters because it doesn't need to
    def testPractice():
      print("Testing practice()...",end = "")
      assert(practice(90) == "A")
      assert(practice(85) == "B")
      assert(practice(70) == "C")
```

```
    print("PASSED.")

testPractice()
```

This function should have given you an assertion error on the line that checks that a score of 70 is equal to a C. Try to change the testPractice function so that the final print statement is reached, and then go back and try to change the practice function instead.

Lesson 9 - Strings

Strings are left for later in the learning curve for one simple reason, they work similar to another data type we'll learn later called lists. Strings, which as a refresher are a "string of characters", that you can do various things with.

You can access a specific part of a string through a concept called **indexing**.

Think of a string being split into little boxes that we can count, like:

"M A R C O"

It's important to note that all counting in programming starts from 0, so the first character, M is at the index 0, and the character O is at the index 4. Despite the string being 5 characters long.

To get a specific character from a string, put brackets next to it with the index you want to access.

Try running this code, and then try writing some of your own! :

```
name = "MARCO"

firstLetter = name[0]

print(firstLetter)
```

Indexing is also great for getting the section of a string, called **slicing,** and is done with string[start : end : step]:

```
school = "Mater"

#end is non-inclusive, so it's actually returning indexes 1-2

print(school[1:4])

#You can also code with default parameters, the colon is neccessary

s = "abcdefgh"
```

```
print(s)

print(s[3:])

print(s[:3])

print(s[:])

#Now with a step

print(s)

print(s[1:7:2])

print(s[1:7:3])
```

You can add two strings together by simply adding them, just like you would numbers.

Example:

```
name = "Marco"

name += " Acea"

print(name)
```

A new line in strings is done using a special character, “\n” :

Example:

```
print("abc\ndef")   # \n is a single newline character

print("""abc

def""")
```

Here are other special characters in Python:

```
print("Double-quote: \"")

print("Backslash: \\")

print("Newline (in brackets): [\n]")

print("Tab (in brackets): [\t]")
```

You can also multiply strings in Python, try testing this code below:

```
print("abc" + "def")

print("abc" * 3)
```

Membership, **this is very important,** you can check if a character or string is inside of another string:

```
print("ring" in "strings")

print("wow" in "amazing!")

print("Yes" in "yes!")

print("" in "No way!")

print("Marco" not in "Mater")

#Challenge: Write a function that checks if a letter is a vowel using membership, and one that checks if it is a consonant.
```

This is a VERY LONG LESSON, BUT ALSO VERY IMPORTANT, make sure to go back and make sure you are comfortable coding using ALL of these skills.

Mini-Lesson 9.5 - Looping over Strings

As explained in class, there are two main methods to loop through strings:

1. Indexing: by using the range function with one parameter (the length of the string), we can effectively loop through the **positions** of the letters in the string:

```
string = "hello world"

for i in range(len(string)):

 print(string[i]) #Notice we're using the index to find the letter, rather than printing the letter itself
```

*Remember, range with one parameter goes from 0 all the way up to (but **not** including) the number given, so in this case, it goes from 0 to the number 10, and stops before it starts 11.*

2. For-each: by instead directly looping through the string, we instead go through the letters themselves:

```
string = "hello world"

for i in string:

 print(i) #Now notice that we don't write range or len, but instead are looking at the letters inside the string
```

For-each loops go through the items inside these things called "iterable objects". They're like lists, which you'll see in Chapter 10, but in this case, the iterable object is the string, and the items are the characters in the string.

Splits

There's a cool command in Python called `split`, and what it does is it splits a string up based on a specific input. You can use it as such:

```
string = "hello world"

string.split()
```

When no parameters are given, it automatically splits a string based on the spaces present in the string (so in the previous example, "hello world" would be broken up into "hello" and "world"). Split actually converts the string into a list (see *Chapter 10*), so it's best to save the result like so:

```
string = "hello world"

splitString = string.split()
```

Although we haven't gone into lists yet, it does allow us to do something pretty cool in the context of looping through strings:

```
string = "hello world"

for i in string.split():

 print(i)
```

If you type this up into Repl.it really quick or run it in Pyzo, you'll see that it lets us loop through the words in string, ignoring spaces! This is important if you want to parse a sentence for a specific phrase or word. Another example, this time with commas:

```
string = "apple, banana, coconut"
```

```
for i in string.split(", "):

 print(i)
```

Lesson 10 - Lists

This is the last data type we'll be covering, lists are a way to group up all the other types, like integers and strings. Let's look at some examples to see it first hand. They have a lot of similar attributes to strings.

To create a list:

```
x = []

#This list can hold any type of data

x = ['hi',3,True]
```

You can check for membership in list:

```
a = [ 2, 3, 5, 2, 6, 2, 2, 7 ]

print("a      =", a)

print("2 in a =", (2 in a))
```

You can add elements to a list:

```
a = [ 2, 3 ]

a.append(7)

print(a)
```

You can add two lists together to make a larger one:

```
a = [ 2, 3 ]

a.extend([ 17, 19 ])

print(a)

a += [1,23,31]

print(a)
```

Removing elements:

```
a = [ 2, 3, 5, 3, 7, 6, 5, 11, 13 ]
print("a =", a)
a.remove(5)
print("After a.remove(5), a=", a)
```

Removing elements at a specific index:

```
a = [ 2, 3, 4, 5, 6, 7, 8 ]
print("a =", a)


item = a.pop(3)
print("After item = a.pop(3)")
print("   item =", item)
print("   a =", a)
# Remove last item with list.pop()
item = a.pop()
print("After item = a.pop()")
print("   item =", item)
print("   a =", a)
```

Looping through a list:

```
a = [ 2, 3, 5, 7 ]
print("Here are the items in a:")
```

```
for item in a:
    print(item)
```

2-D Lists

Want lists inside a list? This is called a 2-D list.

2 -D lists are created by simply using lists as the elements of a list, as such:

```
exampleList = [["list", "1"],["list", "2"],["list", "3"]]
```

This doesn't look that useful, does it? Here's a much more common and relevant example:

```
grid =[[0,0,0,0,0,0,0,0,0,0,0,0],
       [0,1,1,1,0,0,0,0,1,1,1,0],
       [0,1,0,0,0,0,0,0,0,0,1,0],
       [0,1,0,0,0,0,0,0,0,0,1,0],
       [0,0,0,0,0,1,1,0,0,0,0,0],
       [0,0,0,0,0,1,1,0,0,0,0,0],
       [0,1,0,0,0,0,0,0,0,0,1,0],
       [0,1,0,0,0,0,0,0,0,0,1,0],
       [0,1,1,1,0,0,0,0,1,1,1,0],
       [0,0,0,0,0,0,0,0,0,0,0,0]]
```

If you want to copy it into Repl.it to make sure its a valid 2D list, go ahead. It maintains its structure over multiple lines because of the surrounding brackets (mind the commas at the end of every line except the last).

As you may already imagine, using lists this way is much more powerful than having one long array with all the information in an unclustered manner. You don't have to use grids or just strings; you can use lists to contain information about specific variables, objects, etc.

One important thing to note about 2D lists is that if you want to access items within them, the notation for indexing items is slightly different.

Given the following list, how would you try and call the element “banana”?

```
food = [["pork","beef","bacon"],
        ["apple","banana","coconut"],
        ["asparagus","broccoli","carrot"],
        ["pastelitos","croquetas","pan con lechon"]]
```

In this case, you would still call the array, but since banana is the second element in the second array, it would be `food[2][2]`. This means that to iterate through a 2D list, you would need two for loops to run through element. Assuming that the 2D list `food` remains the same, we would use the following for loops to print every item individually (we’ll show this using for loops with index numbers and for each loops):

```
for i in range(len(food)):
 for j in range(len(food[i])):
   print(food[i][j])


for category in food:
 for ingredient in category:
   print(ingredient)
```

Notice that in the index version, we are using the length of each sub array to iterate through the items in the array; in the for each loop version, we use the name of the item we are iterating through, so instead of looping through food, we instead loop through the category that we created.

Anything else you might want to know regarding lists can be found in this link here along with examples!

http://www.cs.cmu.edu/~112/notes/notes-1d-lists.html

Mini - Lesson 10.5 - Tuples

Tuples are like lists, but they can't be changed, so whatever they start as is what they end as.

To write tuples into your code:

```
t = (1, 2, 3)

print(type(t), len(t), t)
```

You can also assign multiple values to a tuple of variables:

```
(x, y) = (1, 2)

print(x)

print(y)

(x, y) = (y, x)

print(x)

print(y)
```

Lesson 11 - Graphics

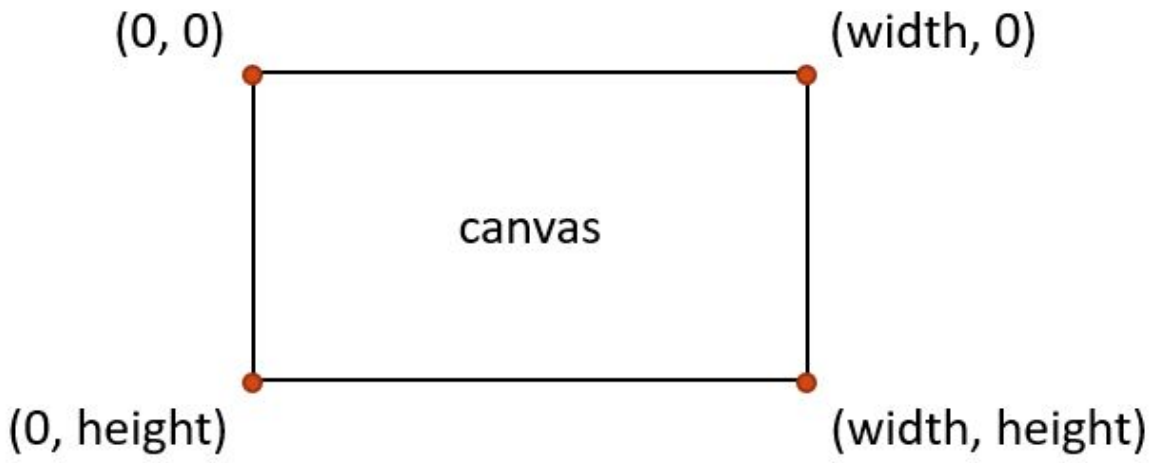
Finally now that you everything you'll need to know about manipulating data, it's actually time to start drawing things with that data. Enter, tkinter, a module (basically code talk for a file of functions someone else wrote but that you can use) that allows you to draw any sort of shape or polygon onto a "canvas". You can also add .gif files, but they have to be specifically one frame (even we don't know why).

This next code, taken from the 15-112 website, is the starter code that you will always need to get a blank canvas onto your screen. Don't worry about what the runDrawing function does, you only have to worry about writing your code in the draw function.

```
from tkinter import *

def draw(canvas, width, height):

   pass # replace with your drawing code!

def runDrawing(width=300, height=300):

   root = Tk()
```

```
    root.resizable(width=False, height=False) # prevents resizing
window
    canvas = Canvas(root, width=width, height=height)
    canvas.configure(bd=0, highlightthickness=0)
    canvas.pack()
    draw(canvas, width, height)
    root.mainloop()
    print("bye!")
runDrawing(400, 200)
```

Coordinates in programing are a little weird, they start from the top left corner at (0,0).



*Taken from the 15-112 Website

There are several different functions you can use within tkinter to draw, here are the ones we think you can/will/should use:

```
#The parameters for this function are as follows
#(top-left x-coordinate, top-left y-coor, bottom-right #x-coor, and
#bottom-right y-coor)
```

```
    #To add color just add fill = name_of_the_color
    canvas.create_rectangle(0,0,150,150)
    canvas.create_rectangle(  0,   0, 150, 150, fill="yellow")
    # ovals provide the coordinates of the bounding box
    canvas.create_oval(100, 50, 300, 150, fill="yellow")
    # polygons and lines provide the (x,y) coordinates of each point
    # polygons must have 3+ points; lines must have 2+
    canvas.create_polygon(100,30,200,50,300,30,200,10, fill="green")
    # text provides a single (x,y) point, then anchors the text there
    # text also requires the text, and can have a font
    canvas.create_text(200, 100, text="Amazing!",\
                       fill="purple", font="Helvetica 26 bold underline")
```

Definitely try these practice problems to get a feel for drawing with tkinter,

http://www.cs.cmu.edu/~112/notes/week2-practice.html

Once you get a hang of drawing stuff, you can change images on the screen depending on the keys, and position of your mouse. This is called event-based animation. The possibilities are endless with this kind of programming because you can create buttons, create game modes, and change the overall flow of what you're drawing.

Here's the starter code you'll always need, don't worry about reading it we'll it explain it by sections:

```
    # Basic Animation Framework
    from tkinter import *
    ####################################
    # customize these functions
    ####################################
```

```
def init(data):
   # load data.xyz as appropriate
   pass
def mousePressed(event, data):
   # use event.x and event.y
   pass
def keyPressed(event, data):
   # use event.char and event.keysym
   pass
def redrawAll(canvas, data):
   # draw in canvas
   pass
####################################
# use the run function as-is
####################################
def run(width=300, height=300):
   def redrawAllWrapper(canvas, data):
       canvas.delete(ALL)
       canvas.create_rectangle(0, 0, data.width, data.height,
                               fill='white', width=0)
       redrawAll(canvas, data)
       canvas.update()
   def mousePressedWrapper(event, canvas, data):
```

```
        mousePressed(event, data)
        redrawAllWrapper(canvas, data)
    def keyPressedWrapper(event, canvas, data):
        keyPressed(event, data)
        redrawAllWrapper(canvas, data)
    # Set up data and call init
    class Struct(object): pass
    data = Struct()
    data.width = width
    data.height = height
    root = Tk()
    root.resizable(width=False, height=False) # prevents resizing
window
    init(data)
    # create the root and the canvas
    canvas = Canvas(root, width=data.width, height=data.height)
    canvas.configure(bd=0, highlightthickness=0)
    canvas.pack()
    # set up events
    root.bind("<Button-1>", lambda event:
                            mousePressedWrapper(event, canvas, data))
    root.bind("<Key>", lambda event:
                            keyPressedWrapper(event, canvas, data))
```

```
    redrawAll(canvas, data)
    # and launch the app
    root.mainloop()  # blocks until window is closed
    print("bye!")
run(400, 200)
```

The **init(data)** function is where all the important information regarding the image is written down, there will be a file attached later where you'll see what we mean by that, but for the most part variables are declared by writing "data.variable_name = blah_blah_blah" .

Here's an example:

```
def init(data):
    # data comes preset with width and height, from the run function
    data.circleSize = min(data.width, data.height) / 10
    data.circleX = data.width/2
    data.circleY = data.height/2
```

The **mousePressed(data,event)** function can access the x and y values of your mouse on the canvas. Later on we'll discuss what actions are recommended to do in this function (Model, View Controller).

```
def mousePressed(event, data):
    data.circleX = event.x
    data.circleY = event.y
```

The **keyPressed(data,event)** function is what will handle what happens with you data when the user presses a key on the keyboard. You could save each letter? Build a paragraph? Pause a game? Move a player? The possibilities are endless.

In this function, **event.char returns the character a user pressed.** View number 8 on the link below to get a better idea of what we mean!

http://www.cs.cmu.edu/~112/notes/notes-animations-part1.html#keyPressed

Finally, the **redrawAll(canvas,data)** function is where you will use all the data you've manipulated to draw out the figures you want to create! This may be the coordinates of a square, circle, a counter, anything you can imagine and redraw on tkinter!

Check out some examples of redrawAll being used here:

http://www.cs.cmu.edu/~112/notes/notes-animations-part1.html#redrawAll

Now that you know how to code

You've now learning you need to know to make a functioning game like tetris! When we took our intro to programming course, around this time we were tasked with writing the code for Tetris! Fear not, we were guided through the process and we encourage you to try and do it yourself too!

To make Tetris follow this link!

http://www.krivers.net/15112-f18/notes/notes-tetris/index.html

When you're done making Tetris, as a final project and testament to the things you've learned, try to do Problem 2 from our Animation Week homework! It will be a grind and a half, but feel free to read through the course website to learn all the little tricks you might need to finish the game!

Once you're done with these two stepping stones, you'll be ready to make your own game for the AP Computer Science: Principles Exam!

Acknowledgments

CMU CS Department, 15-112, Fundamentals of Programming and Computer Science

https://www.cs.cmu.edu/~112/index.html