Video 6 : Variables & Data Types

'''
The 1st thing to know is that all Python files end with the extension .py. Then you have to understand that every programming language must have the ability to Accept, Store and Name Data.

You have to be able to receive data from the keyboard, or from other parts of your program and assign a name to that data. This data is either a single value or multiple values that are assigned a name. Data that is assigned a name and that contains data is called a Variable. Python has many different ways to store lists of data which I'll cover later.
'''

```
# It is convenient to assign names to data. If I want to store my age in Python I'd type
my_age = 43

# If I wanted to store my name I'd type
my_name = "Derek"
```

'''
Rules for Naming Variables

Your variables can start with a letter or _ (Underscore)
After the 1st letter you can use numbers such as num_1
You can't put spaces in variable names my_age is ok, but my age is not
It is considered good practice to separate words with underscores (my_age vs. myAge)
'''

'''
Keywords that you can't us for variable names

and, del, from, not, while, as, elif, global, or, with, assert, else, if, pass, yield, break,
except, import, print, class, exec, in, raise, continue, finally, is, return, def, for, lambda,
try
'''

```
print("Hello", my_name)
```

'''
"Hello" is known as a string. The print() function prints out the your screen the values between its parentheses. If you have multiple values separate them with commas.

Data is stored in essentially boxes in your computers memory. The size of the box you assign is referred to as a data type. If you want to store values with decimal places you store that data in a float data type. If you want to store a series of characters, numbers, etc. you store in a string data type.

'''

'''
Strings

A String is a data type that starts and ends with a “ ‘ or ‘” and contains letters, numbers and other characters. If you find that you want to use a double quote inside of a String proceed it with a backslash like this.
'''

print("\"We never really grow up, we only learn how to act in public\" - Bryan White")

'''
Other Escape Sequences

Newline : \n
Backslash : \\
Single Quote : \’
Backspace : \b
Tab : \t
'''

'''
There are 3 main number types in Python. Integers, floats and complex numbers. I’ll cover complex numbers later.

Integers are values that don’t have decimal values. 3, 8, 100000 are integers. Floats contain decimal values. Pi for example is a float.

There is no maximum value for an integer, as long as you have enough memory. You can however get a practical maximum size with this. Note that you’ll have to import the sys module for this code to work. A module provides prewritten code you can use in your program.
'''

import sys

print(sys.maxsize)

# Maximum size of a float
print(sys.float_info.max)

'''
Please note however that errors can occur when using floats. This is true with all programming languages. When you create a variable a specific amount of space is set aside. If you create a value larger then that space allows errors can creep in. For example
'''

f1 = 1.1111111111111111
f2 = 1.1111111111111111
f3 = f1 + f2
print(f3)

As you can see floats are accurate to 15 digits. Later I’ll introduce data types with more accuracy.

Here is an example of a complex number. I’ll cover them in more detail later.

cn_1 = 5 + 6j

```
# A boolean data type can have either a value of True or False. You'll see how extremely valuable they are later.

can_vote = True

# Python is dynamically typed. What that means is a variables data type is determined by the value you assign to it. This is different from other languages. A variables value can also be changed even if that may sometimes not make sense.

my_age = 43
my_age = "Dog"

# Casting allows you to convert from one type to another. Here is how you cast to the different types. I'll use the type() function to display the new data type for each variable.

# float to int
print("Cast ", type(int(5.4)))
# float to string
print("Cast 2 ", type(str(5.4)))
# unicode character to string
print("Cast 3 ", type(chr(97)))
# character to unicode
print("Cast 4 ", type(ord('a')))
# integer to float
print("Cast 5 ", type(float(2)))

'''
I'm a multi-line comment
'''

# Variable names are case sensitive. For example Age is not the same as age.

age = 2
Age = 3

# Make sure you are casting to the correct data type when working with variables. Also make sure that you surround calculations with parentheses when they produce a single value

num_1 = "1"
num_2 = "2"
print("1 + 2 =", (int(num_1) + int(num_2)))

# That's all for now. In the next video we'll learn about accepting user input and performing math calculations. Please take the quiz below to reenforce what you've learned.
```

VIDEO 7 : User Input & Math Functions

```
'''
It is extremely important to be able to except input from users and Python makes it easy. The
input() function displays a message and then assigns the users input up until they hit return (is-
sue a newline) to a variable. That input is always a string so be prepared to cast if needed.
'''

name = input("What is your name : ")
print("Hello", name)

# You can also except 2 or more values with one input() function. Here we will ask for and then
add together 2 numbers.

num_1, num_2 = input("Enter 2 Numbers : ").split()
# Convert strings into regular numbers (integers)
num_1 = int(num_1)
num_2 = int(num_2)
# Add the values entered and store in sum
sum_1 = num_1 + num_2

# Subtract the values and store in difference
difference = num_1 - num_2

# Multiply the values and store in product
product = num_1 * num_2

# Divide the values and store in quotient
quotient = num_1 / num_2

# Use modulus on the values to find the remainder
remainder = num_1 % num_2

# The format() function matches up values found between the parentheses that follow the key-
word format with the {} (Curly Brackets) that are in the string of the print statement.
print("{} + {} = {}".format(num_1, num_2, sum_1))
print("{} - {} = {}".format(num_1, num_2, difference))
print("{} * {} = {}".format(num_1, num_2, product))
print("{} / {} = {}".format(num_1, num_2, quotient))
print("{} % {} = {}".format(num_1, num_2, remainder))

''' PROBLEM 1
You've learn quite a bit of Python, so I want you to test your knowledge. (Don't look at the solu-
tion that follows below without giving it a try) I want you to write a program that:

Asks the user to input the number of miles
You'll convert miles to kilometers (kilometers = miles * 1.60934)
Then print this for example 5 miles equals 8.0467 kilometers
'''

# Solution

# Ask the user to input miles and assign it to the miles variable
```

```
miles = input('Enter Miles ')

# Convert from string to integer
miles = int(miles)

# Perform calculation by multiplying 1.60934 times miles
kilometers = miles * 1.60934

# Print results using format()
print("{} miles equals {} kilometers".format(miles, kilometers))

''' The Math Module

I'll end this tutorial by providing many of the powerful math module functions Python provides.
A module is a file that contains a bunch of prewritten code. You're technically making a module
right now. I'll cover them in more depth later.
'''

import math
# Because you used import you access methods by referencing the module
print("ceil(4.4) = ", math.ceil(4.4))
print("floor(4.4) = ", math.floor(4.4))
print("fabs(-4.4) = ", math.fabs(-4.4))

# Factorial = 1 * 2 * 3 * 4
print("factorial(4) = ", math.factorial(4))

# Return remainder of division
print("fmod(5,4) = ", math.fmod(5,4))

# Receive a float and return an int
print("trunc(4.2) = ", math.trunc(4.2))

# Return x^y
print("pow(2,2) = ", math.pow(2,2))

# Return the square root
print("sqrt(4) = ", math.sqrt(4))

# Special values
print("math.e = ", math.e)
print("math.pi = ", math.pi)

# Return e^x
print("exp(4) = ", math.factorial(4))

# Return the natural logarithm e * e * e ~= 20 so log(20) tells
# you that e^3 ~= 20
print("log(20) = ", math.log(20))

# You can define the base and 10^3 = 1000
print("log(1000,10) = ", math.log(1000,10))

# You can also use base 10 like this
```

```
print("log10(1000) = ", math.log10(1000))

# We have the following trig functions
# sin, cos, tan, asin, acos, atan, atan2, asinh, acosh,
# atanh, sinh, cosh, tanh
# They follow this format
print("sin(0) ", math.sin(0))

# Convert radians to degrees and vice versa
print("degrees(1.5708) = ", math.degrees(1.5708))
print("radians(90) = ", math.radians(90))
```

VIDEO 8 : CONDITIONAL OPERATORS

'''
Every programming language has the ability to Conditionally Do One Thing or Another. We execute different code depending on different conditions using the keywords if, else and elif. Elif is a shortened way of saying else if.

It is easy to understand why conditional operators. Consider if you went to a restaurant and you were asked whether you want a Coke or a Pepsi. Based on your decision you would then be provided with your choice.
'''

'''
We will use conditional and logical operators in our conditions. Here are the conditional operators :

> : Greater than
< : Less than
>= : Greater than or equal to
<= : Less than or equal to
== : Equal to
!= : Not equal to
'''

```
drink = input("Pick One (Coke or Pepsi) : ")
if drink == "Coke":
    print("Here is your Coke")
elif drink == "Pepsi":
    print("Here is your Pepsi")
else:
    print("Here is your water")
```

Note that if, elif, and else are followed by a condition and then a colon. The code that follows is indented to show what code should be executed depending on the condition. It is important that the lines be indented exactly the same amount on each line.

''' PROBLEM
Taking what you have learned about conditional operators and previous videos, I want you to make a calculator. You'll accept 2 numbers separated by an operator. You'll then use conditional operators to determine what calculation to make. Here is sample output to model :

Enter Calculation : 5 * 6
5 * 6 = 30

Give it a go and see if you can solve the problem. Feel free to use the previous code covered in past videos if you need help. If you don't solve it, don't worry. The goal is only to get you to think in new ways and to understand the final solution.
'''

SOLUTION

```
# Store the user input of 2 numbers and an operator
num_1, operator, num_2 = input('Enter Calculation: ').split()
```

```
# Convert strings into integers
num_1 = int(num_1)
num_2 = int(num_2)

# If, else if (elif) and else execute different code depending on a condition
if operator == "+":
    print("{} + {} = {}".format(num_1, num_2, num_1 + num_2))

# If the 1st condition wasn't true check if this one is
elif operator == "-":
    print("{} - {} = {}".format(num_1, num_2, num_1 - num_2))
elif operator == "*":
    print("{} * {} = {}".format(num_1, num_2, num_1 * num_2))
elif operator == "/":
    print("{} / {} = {}".format(num_1, num_2, num_1 / num_2))

# If none of the above conditions were true then execute this by default
else:
    print("Use either + - * or / next time")

'''
Logical operators can be used to combine conditions. The logical operators are :

and : If both are true it returns true
or : If either are true it returns true
not : Converts true into false and vice versa

I'll now write a program that will determine whether a birthday is important or not. I'll use the
following criteria to determine that.

1 - 18 -> Important
21, 50, > 65 -> Important
All others -> Not Important
'''

# Ask for the users age and cast to an integer
age = int(input("Enter Age : "))

# If age is both greater than or equal to 1 and less than or equal to 18 it is true
if (age >= 1) and (age <= 18):
    print("Important Birthday")

# If age is either 21 or 50 then it is true
elif (age == 21) or (age == 50):
    print("Important Birthday")

# We check if age is less than 65 and then convert true to false or vice versa
# This is the same as if we put age > 65
elif not(age < 65):
    print("Important Birthday")
else:
    print("Sorry Not Important")
```

''' PROBLEM
It's time for you to solve yet another problem. This time we'll determine what grade someone should go to depending on their age. Here is my criteria for determining grade :

1. If age 5 "Go to Kindergarten"
2. Ages 6 through 17 goes to grades 1 through 12 "Go to Grade 6"
3. If age is greater then 17 then say "Go to College"

Here is sample output :

Enter age : 6
Go to Grade 6

Try to complete this with 10 or less lines of code. Feel free to use previous code covered.
'''

SOLUTION

```
# Ask for the age
age = int(input("Enter age: "))

# Handle if age < 5
if age < 5:
    print("Too Young for School")

# Special output just for age 5
elif age == 5:
    print("Go to Kindergarten")

# Since a number is the result for ages 6 - 17 we can check them all
# with 1 condition
# Use calculation to limit the conditions checked
elif (age > 5) and (age <= 17):
    grade = age - 5
    print("Go to Grade {}".format(grade))
# Handle everyone else
else:
    print("Go to College")
```

Ternary Operator

The ternary operator is used to assign one value or another based on a condition. It follows this format condition_true if condition else condition_false

Here is sample code that determines if someone can vote based on their age.

```
age = int(input("What is your age? "))
can_vote = True if age >= 18 else False
print("You can vote :", can_vote)
```

VIDEO 9 : For & Range

'''
Another thing every programming language must do is to provide a way to perform the same action multiple times. The keyword for is one one way in which you can loop through code while executing repeatedly.

For as long as you have more hamburger to eat, keep eating that hamburger.

For loops come in many forms.

You can loop through a list of values. That list is surrounded with square brackets. Each time through the list the next value will be assigned to the variable I in this example.
'''

```
for i in [2, 4, 6, 8, 10]:
    print("i = ", i)
```

'''
We can also have the keyword range define our list for us. range(10) will create a list starting at 0 and go up to, but not include 10, the number passed into it.

We can now combine range with a for loop to print the numbers 0 through 4 like so
'''

```
for i in range(5):
    print("i = ", i)

# We can also define the starting and ending values with range. This time we'll print the values 2 through 4.

for i in range(2, 5):
    print("i = ", i)
```

'''
Now before you solve another problem I want to teach you how to test if a number is odd or even. If you divide any even number by 2 it will not have a remainder. The modulus operator provides the remainder of a division as I covered previously.

So if I % 2 == 0 we know that I is an even value.
'''

```
i = 6
print("Is 6 even :", ((i % 2) == 0))
```

''' PROBLEM
Use the knowledge you have gained to print out all odd numbers from 1 to 20. Feel free to look at everything on this page to help.
'''

```
# SOLUTION

# Use for to loop through the list from 1 to 21
```

```
for i in range(1, 21):
    # Use modulus to check that the result is NOT EQUAL to 0
    # Print the odds
    if (i % 2) != 0:
        print("i = ", i)

# As you recall, floats are values that have decimal values. We can convert string input into a float like this

your_float = input("Enter a float : ")
your_float = float(your_float)

# We can also use format with print to define the number of decimals displayed in output. If you wanted to show 2 decimal values you would use {:.2f} like in this example

print("Rounded to 2 decimals : {:.2f}".format(your_float))

''' PROBLEM
In this problem you will calculate how much money a person will have after investing for 10 years. Compounding interest is the act of reinvesting each years interest payment and then receiving interest on the initial value as well as on interest payments.

Your program will :

1. Have the user enter their investment amount and expected interest
2. Each year their investment will increase by their investment + their investment * the interest rate
3. Print out their earnings after a 10 year period
'''

# SOLUTION

# Ask for money invested + the interest rate
money = input("How much to invest: ")
interest_rate = input("Interest Rate: ")

# Convert value to a float
money = float(money)

# Convert value to a float and round the percentage rate by 2 digits
interest_rate = float(interest_rate) * .01

# Cycle through 10 years using for and range from 0 to 9
for i in range(10):
    # Add the current money in the account + interest earned that year
    money = money + (money * interest_rate)

# Output the results
print("Investment after 10 years: {:.2f}".format(money))

'''
When a computer tries to solve a calculation it follows certain rules. It will for example multiply values next to each other before adding them, no matter what the order is. These rules are known as the Order of Operations. Calculations will occur in this order :
```

1. exponentiation and root extraction
2. multiplication and division
3. addition and subtraction
'''

```
print("2 + (3 * 4) = ", (2 + (3 * 4)))
print("(2 + 3) * 4 = ", ((2 + 3) * 4))

# That's it for this video. In the next video I'll cover While Loops, Random Values, Break, Continue, and I'll provide more problems for you to solve.
```

VIDEO 10 : WHILE LOOP, BREAK & CONTINUE

Previously I talked about how you can loop with for. We can also continue looping as long as a condition is true with a while loop. While loops are used when you don't know how many times you will have to loop.

Here we'll generate a random number with the random module and randrange(). We will then use the while loop to guess the random value and output it.

CODE

```
# We can use the random module to generate random numbers
import random

# Generate a random integer between 1 and 50
rand_num = random.randrange(1, 51)

# The value we increment in the while loop is defined before the loop
i = 1

# Define the condition that while true we will continue looping
while i != rand_num:

    # You must increment your iterator inside the while loop
    i += 1

# Outside of the while loop when we stop adding whitespace
print("The random value is : ", rand_num)
```

Break & Continue

Break and continue are very useful. Continue stops executing the code that remains in the loop and jumps back to the top. While break ends execution and jumps directly to the code that lies immediately outside of the loop.

Here we'll cycle from 0 to 20 with a while loop. If a number is even will use continue to skip printing it. If it is odd we'll print it. We'll then end execution with break if the value ever reaches 15.

CODE

```
i = 1
 while i <= 20:

    # If a number is even don't print it
    if (i % 2) == 0:
        i += 1
        continue

    # If i equals 15 stop looping
    if i == 15:
        break
```

```
    # Print the odds
    print("Odd : ", i)

    # Increment i
    i += 1
```

Python Problem for you to Solve

For this problem I want you to draw a pine tree after asking the user for the number of rows. This problem is the most difficult you have had so far, but it will teach a lot. Feel free to use any resources online to solve it.

Here is the sample program

```
How tall is the tree : 5
      #
     ###
    #####
   #######
#########
      #
```

Here are some additional tips to help you solve the problem.

Tip 1

You should use a while loop and 3 for loops.

Tip 2

I know that this is the number of spaces and hashes for the tree
4 - 1
3 - 3
2 - 5
1 - 7
0 - 9
Spaces before stump = Spaces before top

TIP 3

You will need to do the following in your program :

1. Decrement spaces by one each time through the loop
2. Increment the hashes by 2 each time through the loop
3. Save spaces to the stump by calculating tree height - 1
4. Decrement from tree height until it equals 0
5. Print spaces and then hashes for each row
6. Print stump spaces and then 1 hash

Solution

CODE

```
# Get the number of rows for the tree
tree_height = input("How tall is the tree : ")

# Convert into an integer
tree_height = int(tree_height)

# Get the starting spaces for the top of the tree
spaces = tree_height - 1

# There is one hash to start that will be incremented
hashes = 1

# Save stump spaces til later
stump_spaces = tree_height - 1

# Makes sure the right number of rows are printed
while tree_height != 0:

    # Print the spaces
    # end="" means a newline won't be added
    for i in range(spaces):
        print(' ', end="")

    # Print the hashes
    for i in range(hashes):
        print('#', end="")

    # Newline after each row is printed
    print()

    # I know from research that spaces is decremented by 1 each time
    spaces -= 1

    # I know from research that hashes is incremented by 2 each time
    hashes += 2

    # Decrement tree height each time to jump out of loop
    tree_height -= 1

# Print the spaces before the stump and then a hash
for i in range(stump_spaces):
    print(' ', end="")

print("#")
```

That's it for todays lesson. Good job if you solved the Christmas tree problem and if you didn't don't worry. Remember the goal is to only get you to think in new ways and to understand the solution.

In the next video we'll cover Exception handling and Accurate Floats.

VIDEO 11 : Exception Handling & Accurate Floats

Nobody wants their programs to crash. We should write code that anticipates the bad things a user may enter to cause a crash and eliminate them. However some times the user may try to access a database that doesn't exist, or they simply refuse to follow instructions. That is where exception handling comes in.

Through exception handling we can catch an error that would normally crash our program and give the user the opportunity to do the right thing.

In this 1st example we'll ask the user to enter a number, but if they refuse we will ask again by catching and solving the error.

We will surround our problematic code with a try block. We will then try to catch the expected error in an except block. If the user does something completely unexpected we will use an except block without defining a specific exception to catch all other exceptions.

CODE

```
# By giving the while a value of True it will cycle until a break is reached
while True:

    # If we expect an error can occur surround potential error with try
    try:
        number = int(input("Please enter a number : "))
        break

    # The code in the except block provides an error message to set things right
    # We can either target a specific error like ValueError
    except ValueError:
        print("You didn't enter a number")

    # We can target all other errors with a default
    except:
        print("An unknown error occurred")

print("Thank you for entering a number")
```

As we continue we will cover other common exceptions you should know how to catch and solve.

Python Problem for you to Solve

I showed you a new trick you can use with a while loop above. Now I want you to implement a Do While loop in Python using that trick along with break, which you learned about in the last video.

The rules for a Do While loop is that they always execute all of the code at least once. After that 1st loop if the condition is true they will run that code again.

In other programming languages they have the format of

```
do {
```

... Bunch of code ...
} while(condition)

I want you to create a guessing game in which the user must chose a number between 1 and 10 with the following format.

Guess a number between 1 and 10 : 1
Guess a number between 1 and 10 : 3
Guess a number between 1 and 10 : 5
Guess a number between 1 and 10 : 7
You guessed it

Try your best. The goal is to get you to think in new ways and to understand the final solution.
Hint : You'll need a while and break.

Solution

CODE

```
secret_number = 7

while True:
    guess = int(input("Guess a number between 1 and 10 : "))

    if guess == secret_number:
        print("You guessed it")
        break
```

More Accurate Floats

If you are not satisfied with 15 decimals of procession in your floating post calculations then you're in luck.

The decimal module provides for more accurate floating point calculations. With from you can reference methods without the need to reference the module like we had to do with math in a previous video. 28 points of precision by default.

We can also create an alias being D here to avoid conflicts with methods with the same name.

Here is some example code you can use to work with accurate floats.

CODE

```
# from decimal import Decimal as D

sum = D(0)
sum += D("0.01")
sum += D("0.01")
sum += D("0.01")
sum -= D("0.03")

print("Sum = ", sum)

from decimal import *
sum_1 = Decimal(0)
```

```
sum_1 += Decimal("0.0111111111111111")
sum_1 += Decimal("0.0111111111111111")
sum_1 += Decimal("0.0111111111111111")
print("Sum = ", sum_1)
sum_1 -= Decimal("0.0333333333333333")
print("Sum = ", sum_1)
```

That’s all for this video. In the next video I’ll cover strings in detail and of course I’ll have more problems for you to solve.

VIDEO 12 : STRINGS

This tutorial will be very code heavy because I'm going to show you a ton of string functions. Strings are a series of characters between quotes. You can use Single quotes, double quotes or triple quotes.

CODE

```
print(type("3"))
print(type('3'))
print(type('''3'''))
```

You can see the data type for data using type

CODE

```
print(type(3))
print(type(3.14))
```

Each character is stored in a series of boxes labeled with index numbers. You can find out how many characters a string contains.

CODE

```
samp_string = "This is a very important string"
print("Length :", len(samp_string))
```

You can get characters using index numbers starting at 0.

CODE

```
samp_string = "This is a very important string"
print(samp_string[0])

# Get the last character
print(samp_string[-1])
```

SLICE

You can get a block of characters using slice. A slice is where you define what index values you want between 2 brackets.

CODE

```
samp_string = "This is a very important string"
# Get a slice by saying where to start and end
# The 4th index isn't returned
print(samp_string[0:4])

# Get everything starting at an index
print(samp_string[8:])

# More slices
```

```
print("Every Other ", samp_string[0:-1:2])
print("Reverse ", samp_string[::-1])
```

Other random string manipulations you can use

CODE

```
# Join or concatenate strings with +
print("Green " + "Eggs")

# Repeat strings with *
print("Hello " * 5)

# Convert an int into a string
num_string = str(4)
```

You can cycle through each character with for

CODE

```
samp_string = "This is a very important string”
for char in samp_string:
    print(char)
```

You can cycle through characters in pairs. Subtract 1 from the length because length is 1 more then the highest index because strings are 0 indexed. Then use range starting at index 0 through string length and increment by 2 each time through.

CODE

```
samp_string = "This is a very important string”
for i in range(0, len(samp_string)-1, 2):
    print(samp_string[i] + samp_string[i+1])
```

UNICODE

Computers assign characters with a number known as a Unicode A-Z have the numbers 65-90 and a-z 97-122. 2 functions allow you to work with unicodes.

CODE

```
# You can get the Unicode code with ord()
print("A =", ord("A"))

# You can convert from Unicode with chr
print("65 =", chr(65))
```

Shortcut Ways to Perform Math Calculations

Let’s say you want to add val plus 1. You could type out val = val + 1, but there is a shortcut way val_1 += 1

This shortcut can be used for all math operations

```
val_1 -= 5
val_1 *= 3
val_1 /= 2
val_1 %= 6
```

Likewise you can also add one string to another in the same way

```
str_1 += str_2
```

There is another shortcut when you want to just increment or decrement by 1. Instead of val_1 += 1 you can just type val_1++ or val_1—.

Python Problem for you to Solve

Here is another problem you can work through. Remember it isn't important if you don't get it right. Think in new ways, search the internet and the only goal is to understand the solution.

Your code should receive a **uppercase string** and then hide it's meaning by turning it into a string of unicodes. Then it should translate the unicodes back into the original message.

SOLUTION

```
norm_string = input("Enter a string to hide in uppercase: ")

secret_string = ""

# Cycle through each character in the string
for char in norm_string:

    # Store each character code in a new string
    # += is the same as secret_string = secret_string + whatever
    secret_string += str(ord(char))

print("Secret Message :", secret_string)

norm_string = ""

# Cycle through each character code 2 at a time by incrementing by
# 2 each time through since unicodes go from 65 to 90
for i in range(0, len(secret_string)-1, 2):

    # Get the 1st and 2nd for the 2 digit number
    char_code = secret_string[i] + secret_string[i+1]

    # Convert the codes into characters and add them to the new string
    norm_string += chr(int(char_code))

print("Original Message :", norm_string)
```

2nd Python Problem for you to Solve

Now if you solved the previous problem I have another for you. Make the above work with upper and lowercase letters by changing 2 lines of code.

SOLUTION

Add these 2 changes

secret_string += str(ord(char) - 22)

norm_string += chr(int(char_code) + 22)

That's all we will cover on strings this time. In the next video we'll cover even more with strings including more problems for you to solve.

VIDEO 13 : MORE STRING FUNCTIONS

This time we'll cover most of the string methods that you'll ever need, that have not been covered previously. I'll display some examples with a brief explanation which should be self explanatory.

CODE

```
# Strings have many methods we can use beyond what I covered last time
rand_string = "    this is an important string    "

# Delete whitespace on left
rand_string = rand_string.lstrip()

# Delete whitespace on right
rand_string = rand_string.rstrip()

# Delete whitespace on right and left
rand_string = rand_string.strip()

# Capitalize the 1st letter
print(rand_string.capitalize())

# Capitalize every letter
print(rand_string.upper())

# lowercase all letters
print(rand_string.lower())
```

Lists will be covered in detail later, but I want to show them briefly here. This is how you turn a list into a string and then separate each item with a space.

CODE

```
a_list = ["Bunch", "of", "random", "words"]
print(" ".join(a_list))
```

A space was defined as the thing that would separate the items in the last example. Something that is used to separate data is called a delimiter. So if we had "pig, cow, turtle" the comma and space would be the delimiter because it comes between each piece of meaningful data.

This is how we turn a string into a list and print it out.

CODE

```
rand_string = "this is an important string"
a_list_2 = rand_string.split()
```

```
for i in a_list_2:
    print(i)
```

We can find how many times a string occurs in a string

CODE

```
rand_string = “this is an important string”
print("How many is :", rand_string.count(“is"))
```

Get an index for a matching string

CODE

```
rand_string = “this is an important string”
print("Where is string :", rand_string.find(“string"))
```

Replace a string

CODE

```
rand_string = “this is an important string”
print(rand_string.replace("an ", "a kind of "))
```

Python Problem for you to Solve

It’s time to test what you have learned. You will create an acronym generator. The user will enter a string and then convert it to an acronym with uppercase letters like this

Convert to Acronym : Random Access Memory
RAM

Give it a go. I’m sure you can do it.

SOLUTION

```
# Ask for a string
orig_string = input("Convert to Acronym : ")

# Convert the string to all uppercase
orig_string = orig_string.upper()

# Convert the string into a list
list_of_words = orig_string.split()

# Cycle through the list
for word in list_of_words:

    # Get the 1st letter of the word and eliminate the newline
    print(word[0], end="")

print()
```

More String Methods

For our next problem some additional string methods are going to be very useful.

CODE

```
# Returns True if characters are letters or numbers
# Whitespace is false
print("Is z a letter or number :", letter_z.isalnum())

# Returns True if characters are letters
print("Is z a letter :", letter_z.isalpha())

# Returns True if characters are numbers (Floats are False)
print("Is 3 a number :", num_3.isdigit())

# Returns True if all are lowercase
print("Is z a lowercase :", letter_z.islower())

# Returns True if all are uppercase
print("Is z a uppercase :", letter_z.isupper())

# Returns True if all are spaces
print("Is space a space :", a_space.isspace())
```

2nd Python Problem for you to Solve

This problem will really wrack your brain, so don't worry if you can't solve it. Feel free to use all the resources of the internet to try.

We are going to make a Caesar's Cipher. Encryption is super popular so let's take a look at one of the first. Here is what you'll have to program.

Receive a message and then encrypt it by shifting the characters by a requested amount to the right. A becomes D, B becomes E for example. Also decrypt the message back again.

You should check if a character is a letter and if not leave it as its default.

HINTS

1. A-Z have the numbers 65-90 in unicode
2. a-z have the numbers 97-122
3. You get the unicode of a character with ord(yourLetter)
4. You convert from unicode to character with chr(yourNumber)
5. Use isupper() to decided which unicodes to work with
6. Add the key (number of characters to shift) and if bigger or smaller then the unicode for A, Z, a, or z increase or decrease by 26

SOLUTION

```
# Receive the message to encrypt and the number of characters to shift
message = input("Enter your message : ")
key = int(input("How many characters should we shift (1 - 26)"))

# Prepare your secret message
secret_message = ""
```

```
# Cycle through each character in the message
for char in message:

    # If it isn't a letter then keep it as it is in the else below
    if char.isalpha():

        # Get the character code and add the shift amount
        char_code = ord(char)
        char_code += key

        # If uppercase then compare to uppercase unicodes
        if char.isupper():

            # If bigger than Z subtract 26
            if char_code > ord('Z'):
                char_code -= 26

            # If smaller than A add 26
            elif char_code < ord('A'):
                char_code += 26

        # Do the same for lowercase characters
        else:
            if char_code > ord('z'):
                char_code -= 26
            elif char_code < ord('a'):
                char_code += 26

        # Convert from code to letter and add to message
        secret_message += chr(char_code)

    # If not a letter leave the character as is
    else:
        secret_message += char

print("Encrypted :", secret_message)

# To decrypt the only thing that changes is the sign of the key
key = -key

orig_message = ""

for char in secret_message:
    if char.isalpha():
        char_code = ord(char)
        char_code += key

        if char.isupper():
            if char_code > ord('Z'):
                char_code -= 26
            elif char_code < ord('A'):
                char_code += 26
        else:
```

```
            if char_code > ord('z'):
                char_code -= 26
            elif char_code < ord('a'):
                char_code += 26

        orig_message += chr(char_code)

    else:
        orig_message += char

print("Decrypted :", orig_message)
```

Awesome Job if you solved that! Really awesome super job if you understood all of the code! The goal is only to make you think like a programmer and to understand the finished code.

I'll give you a break this time and not force you to do a quiz since the problems were so difficult. I won't be easy on you next time though. In the next video we'll finally start talking about functions.

VIDEO 14 : FUNCTIONS

Functions allow use to reuse code and make the code easier to understand. To create a function type def (define) the function name and then in parentheses a comma separated list of values that function can accept.

This function adds 2 values and returns the sum.

CODE
```
def add_numbers(num_1, num_2):

    # Return returns a value if needed
    return num_1 + num_2

# You call the function by name followed by passing comma
# separated values if needed and a value may or may not be
# returned

print("5 + 4 =", add_numbers(5, 4))
```

Function Local Variables

Any variable defined inside a function is not available outside of that function. For example

CODE
```
def assign_name():
    name = "Doug"

assign_name()

# Throws a NameError
# print(name)
```

Global Variables

You can't change a global variable even if it is passed into a function. That is because a value and not the actual variable is passed to the function.

CODE
```
def change_name(name):

    # Trying to change the global
    name = "Mark"

# A variable defined outside of a function can't be changed
# in the function using the above way
name = "Tom"
```

```
# Try to change the value
change_name(name)

# Prints Tom even though the function tries to change it
print(name)
```

If you want to change the value pass it back

CODE

```
def change_name_2():
    return "Mark"

name = change_name_2()
print(name)
```

You can also use the global keyword to change it.

CODE

```
gbl_name = "Sally"

def change_name_3():
    global gbl_name
    gbl_name = "Sammy"

change_name_3()
print(gbl_name)
```

If you don’t return a value a function will return none.

CODE

```
def get_sum(num1, num2):
    sum = num1 + num2

print(get_sum(5, 4))
```

MAKE A is_float FUNCTION

There is no way to check if a string contains a float so let's make one by defining our own function.

```
def is_float(str_val):
    try:

        # If the string isn't a float float() will throw a
        # ValueError
        float(str_val)

        # If there is a value you want to return use return
        return True
    except ValueError:
        return False

pi = 3.14
print("Is Pi a Float :", is_float(pi))
```

Python Problem for you to Solve

In this problem you'll receive an algebraic equation and solve for x. Know that x will always be the 1st value received and you only will deal with addition. Here is a sample of calling the function and then the output.

```
print(solve_eq("x + 4 = 9"))
x = 5
```

Solution

```
def solve_eq(equation):
    x, add, num1, equal, num2 = equation.split()

    # Convert the strings into ints
    num1, num2 = int(num1), int(num2)

    # Convert the result into a string and join (concatenate)
    # it to the string "x = "
    return "x = " + str(num2 - num1)

print(solve_eq("x + 4 = 9"))
```

That's it for this video. In the next video I'll show you how to return and receive multiple values. We'll also calculate primes, calculate areas for different shapes and talk about main().

VIDEO 15 : FUNCTIONS 2

In this video I'll show you how to return and receive multiple values. We'll also calculate primes, calculate areas for different shapes and talk about main().

To return multiple values just separate values returned with commas.

CODE

```
def mult_divide(num1, num2):
    return (num1 * num2), (num1 / num2)

mult, divide = mult_divide(5, 4)

print("5 * 4 =", mult)
print("5 / 4 =", divide)
```

Return a List of Primes

A prime can only be divided by 1 and itself. 5 is prime because 1 and 5 are its only positive factors. 6 is a composite because it is divisible by 1, 2, 3 and 6.

We'll receive a request for primes up to the input value. We'll then use a for loop and check if modulus == 0 for every value up to the number to check. If modulus == 0 that means the number isn't prime.

CODE

```
def isprime(num):
    # This for loop cycles through primes from 2 to
    # the value to check
    for i in range(2, num):

        # If any division has no remainder we know it
        # isn't a prime number
        if (num % i) == 0:
            return False
    return True


def getPrimes(max_number):

    # Create a list to hold primes
    list_of_primes = []

    # This for loop cycles through primes from 2 to
    # the maximum value requested
    for num1 in range(2, max_number):
```

```
        if isprime(num1):
            list_of_primes.append(num1)

    return list_of_primes

max_num_to_check = int(input("Search for Primes up to : "))

list_of_primes = getPrimes(max_num_to_check)

for prime in list_of_primes:
    print(prime)
```

Unknown Number of Arguments

We can receive an unknown number of arguments using the splat (*) operator

CODE

```
def sumAll(*args):
    sum = 0

    for i in args:
        sum += i
        return sum

print("Sum :", sumAll(1,2,3,4))
```

Route to Different Functions & Main()

Here we will route to different functions depending on what type of shape we want to get the area for. We will also use a main function for the first time.

CODE

```
Import math

# This routes to the correct area function
# The name of the value passed doesn't have to match
def get_area(shape):

    # Switch to lowercase for easy comparison
    shape = shape.lower()

    if shape == "rectangle":
        rectangle_area()
    elif shape == "circle":
        circle_area()
    else:
        print("Please enter rectangle or circle")

# Create function that calculates the rectangle area
def rectangle_area():
    length = float(input("Enter the length : "))
```

```
    width = float(input("Enter the width : "))

    area = length * width

    print("The area of the rectangle is", area)


# Create function that calculates the circle area
def circle_area():
    radius = float(input("Enter the radius : "))

    area = math.pi * (math.pow(radius, 2))

    # Format the output to 2 decimal places
    print("The area of the circle is {:.2f}".format(area))

# We often place our main programming logic in a function called main
# We create it this way

def main():

    # Our program will calculate the area for rectangles or circles

    # Without functions we'd have to create a giant list of ifs, elifs

    # Ask the user what shape they have
    shape_type = input("Get area for what shape : ")

    # Call a function that will route to the correct function
    get_area(shape_type)

    # Because of functions it is very easy to see what is happening
    # For more detail just refer to the very short specific functions

# All of the function definitions are ignored and this calls for main()
# to execute when the program starts

main()
```

For additional homework you can add the ability to calculate area for parallelograms, rhombus, triangles, and trapezoids.

That’s it for now. Get ready for the next video because we’ll cover lists. We’ll look at numerous list functions, how to sort lists with the Bubble Sort and much more.

VIDEO 16 : LISTS

In this video we'll look at numerous list functions, how to sort lists with the Bubble Sort and much more.

With lists we can refer to groups of data with 1 name. Each item in the list corresponds to a number (index) just like how people have identification numbers. By default the 1st item in a list has the index 0.

Ex. [0 : "string"] [1 : 1.234] [2 : 28] [3 : "c"]

Python lists can grow in size and can contain data of any type. An awesome thing about lists is that you can use many of the same functions with them that you used with strings.

CODE

```
rand_list = ["string", 1.234, 28]

# Create a list with range
one_to_ten = list(range(10))

# Combine lists
rand_list = rand_list + one_to_ten

# Get the 1st item with an index
print(rand_list[0])

# Get the length
print("List Length :", len(rand_list))

# Slice a list to get 1st 3 items
first3 = rand_list[0:3]

# Cycle through the list and print the index
for i in first3:
    print("{} : {}".format(first3.index(i), i))

# You can repeat a list item with *
print(first3[0] * 3)

# You can see if a list contains an item
print("string" in first3)

# You can get the index of a matching item
print("Index of string :", first3.index("string"))

# Find out how many times an item is in the list
print("How many strings :", first3.count("string"))

# You can change a list item
first3[0] = "New String"
```

```
for i in first3:
    print("{} : {}".format(first3.index(i), i))

# Append a value to the end of a list
first3.append("Another")
```

Python Problem for you to Solve

For this problem generate a random list of 5 values between 1 and 9

Solution

CODE

```
Import random
num_list = []
for i in range(5):
    num_list.append(random.randrange(1, 9))
```

Bubble Sort

The Bubble sort is a way to sort a list. It works this way :

1. An outer loop decreases in size each time
2. The goal is to have the largest number at the end of the list when the outer loop completes 1 cycle
3. The inner loop starts comparing indexes at the beginning of the loop
4. Check if list[Index] > list[Index + 1]
5. If so swap the index values
6. When the inner loop completes the largest number is at the end of the list
7. Decrement the outer loop by 1

Here is a bubble sort in code

CODE

```
# Create the value that will decrement for the outer loop
# Its value is the last index in the list
i = len(num_list) - 1

while i > 1:

    j = 0

    while j < i:

        # Tracks the comparison of index values
        print("\nIs {} > {}".format(num_list[j], num_list[j+1]))
        print()

        # If the value on the left is bigger switch values
        if num_list[j] > num_list[j+1]:

            print("Switch")
```

```
            temp = num_list[j]
            num_list[j] = num_list[j + 1]
            num_list[j + 1] = temp

        else:
            print("Don't Switch")

        j += 1

        # Track changes to the list
        for k in num_list:
            print(k, end=", ")
        print()

    print("END OF ROUND")

    i -= 1

for k in num_list:
    print(k, end=", ")
print()
```

More List Functions

CODE

```
Import random
num_list = []
for i in range(5):
    num_list.append(random.randrange(1, 9))

# Sort a list
num_list.sort()

# Reverse a list
num_list.reverse()

for k in num_list:
    print(k, end=", ")
print()

# Insert value at index insert(index, value)
num_list.insert(5, 10)

# Delete first occurrence of value
num_list.remove(10)

for k in num_list:
    print(k, end=", ")
print()

# Remove item at index
num_list.pop(2)
```

```
for k in num_list:
    print(k, end=", ")
print()
```

That’s all for this video. In the next video I’ll cover List Comprehensions, Multidimensional Lists and you’ll have to solve another problem.

VIDEO 17 : LISTS 2

In this video we'll cover List Comprehensions, Multidimensional Lists and you'll have to solve another problem.

List Comprehensions

You can construct lists in interesting ways using list comprehensions. You can do this by performing an operation on each item in the list.

CODE

```
Import math
# Create a list of even values
even_list = [i*2 for i in range(10)]

for k in even_list:
    print(k, end=", ")
print()

# List of lists containing values to the power of
# 2, 3, 4
num_list = [1,2,3,4,5]

list_of_values = [[math.pow(m, 2), math.pow(m, 3), math.pow(m, 4)]
                  for m in num_list]

for k in list_of_values:
    print(k)
print()

# Create a 10 x 10 list
multi_d_list = [[0] * 10 for i in range(10)]

# Change a value in the multidimensional list
multi_d_list[0][1] = 10

# Get the 2nd item in the 1st list
# It may help to think of it as the 2nd item in the 1st row
print(multi_d_list[0][1])

# Get the 2nd item in the 2nd list
print(multi_d_list[1][1])
```

Multidimensional Lists

Multidimensional list are tables of data that spans across rows and columns. Here I'll show how indexes work with a multidimensional list.

CODE

```
list_table = [[0] * 10 for i in range(10)]

for i in range(10):

    for j in range(10):
        list_table[i][j] = "{} : {}".format(i, j)

for i in range(10):

    for j in range(10):
        print(list_table[i][j], end=" || ")
    print()
```

Python Problem for you to Solve

With 2 for loops fill the cells in a multidimensional list with a multiplication table using values 1 - 9. This is your goal.

1, 2, 3, 4, 5, 6, 7, 8, 9,
2, 4, 6, 8, 10, 12, 14, 16, 18,
3, 6, 9, 12, 15, 18, 21, 24, 27,
4, 8, 12, 16, 20, 24, 28, 32, 36,
5, 10, 15, 20, 25, 30, 35, 40, 45,
6, 12, 18, 24, 30, 36, 42, 48, 54,
7, 14, 21, 28, 35, 42, 49, 56, 63,
8, 16, 24, 32, 40, 48, 56, 64, 72,
9, 18, 27, 36, 45, 54, 63, 72, 81

Solution

CODE

```
# Create the multidimensional list
mult_table = [[0] * 10 for i in range(10)]

# This will increment for each row
for i in range(1, 10):

    # This will increment for each item in the row
    for j in range(1, 10):

        # Assign the value to the cell
        mult_table[i][j] = i * j

# Output the data in the same way you assigned it
for i in range(1, 10):

    for j in range(1, 10):
        print(mult_table[i][j], end=", ")

    print()
```

That’s it for this video. In the next video we’ll cover Dictionaries.

QUIZ

1. Create even_list which will contain even numbers up till 10? even_list = [i*2 for i in range(10)]
2. Create a multidimensional list list_table with 10 rows and columns? list_table = [[0] * 10 for i in range(10)]

VIDEO 18 : Dictionaries

While lists organize data based on sequential indexes Dictionaries instead use key / value pairs. A key / value pair could be fName : "Derek" where fName is the key and "Derek" is the value. Here is some code to help this make sense.

CODE

```
# Create a Dictionary about me
derek_dict = {"f_name": "Derek", “l_name": "Banas", "address": "123 Main St"}

# Get a value with the key
print("My name :", derek_dict["f_name"])

# Change a value with the key
derek_dict["address"] = "215 North St"

# Dictionaries may not print out in the order created
# since they are unordered
print(derek_dict)

# Add a new key value
derek_dict['city'] = 'Pittsburgh'

# Check if a key exists
print("Is there a city :", "city" in derek_dict)

# Get the list of values
print(derek_dict.values())

# Get the list of keys
print(derek_dict.keys())

# Get the key and value with items()
for k, v in derek_dict.items():
    print(k, v)

# Get gets a value associated with a key or the default
print(derek_dict.get(“m_name", "Not Here"))

# Delete a key value
del derek_dict["f_name"]

# Loop through the dictionary keys
for i in derek_dict:
    print(i)

# Delete all entries
derek_dict.clear()

# List for holding Dictionaries
```

```
employees = []

# Input employee data
f_name, l_name = input("Enter Employee Name : ").split()

employees.append({'f_name': f_name, 'l_name': l_name})

print(employees)
```

Python Problem for you to Solve

Create an array of customer dictionaries and the output should look like this :

```
Enter Customer (Yes/No) : y
Enter Customer Name : Derek Banas
Enter Customer (Yes/No) : y
Enter Customer Name : Sally Smith
Enter Customer (Yes/No) : n
Derek Banas
Sally Smith
```

Solution

```
# Create customer array outside the for so it isn't local
# to the while loop
customers = []

while True:

    # Cut off the 1st letter to cover if the user
    # types a n or y
    create_entry = input("Enter Customer (Yes/No) : ")
    create_entry =  create_entry[0].lower()

    if  create_entry == "n":

        # Leave the while loop when n is entered
        break
    else:

        # Get the customer name by splitting at the space
        f_name, l_name = input("Enter Customer Name : ").split()

        # Add the dictionary to the array
        customers.append({‘f_name': f_name, ‘l_name': l_name})

# Print out customer list
for cust in customers:
    print(cust[‘f_name'], cust[‘l_name'])
```

That’s it for this video. In the next part of this tutorial we will cover recursive functions, which are functions that execute themselves.

VIDEO 19 : RECURSIVE FUNCTIONS

A recursive function is a function that calls itself. You may ask yourself, why would you ever want to do that? Actually certain problems can be solved more easily through recursion.

Every recursive function must contain a condition that stops the process of calling the function to execute. Then we break down solving a problem by by performing multiple simple calculations repetitively versus writing one large block of code. Take calculating a factorial as an example.

Calculating a Factorial

Calculating factorials is commonly done with a recursive
function 3! = 3 * 2 * 1

CODE

```
def factorial(num):

    # Every recursive function must contain a condition
    # when it ceases to call itself
    if num <= 1:
        return 1
    else:

        result = num * factorial(num - 1)
        return result

print(factorial(4))
```

How it works

1st : result = 4 * factorial(3) = 4 * 6 = 24
2nd : result = 3 * factorial(2) = 3 * 2 = 6
3rd : result = 2 * factorial(1) = 2 * 1 = 2

To solve this problem we work down the ladder of calculations up until the factorial function is passed the value of 1. When that occurs during the 3rd pass we can complete our 1st calculation getting the result of 2. We then move the 2 up the ladder to solve the next calculation (3 * 2 = 6). Then the 6 moves up to finish the calculation (4 * 6 = 24)

Python Problem for you to Solve

To test what you have learned I now want you to calculate Fibonacci numbers.

To calculate Fibonacci numbers we sum the 2 previous values to calculate the next item in the list like this 1, 1, 2, 3, 5, 8 ...

The Fibonacci sequence is defined by:
Fn = Fn-1 + Fn-2
Where F0 = 0 and F1 = 1

Here is a sample run through to help

print(fib(3))

1st : result = fib(2) + fib(1) : 2 + 1
2nd : result = (fib(1) + fib(0)) + (fib(0)) : 1 + 0
3rd : result = fib(2) + fib(1)

print(fib(4))

1st : result = fib(3) + fib(2) : 3 + 2
2nd : result = (fib(2) + fib(1)) + (fib(1) + fib(0)) : 2 + 1
3rd : result = (fib(1) + fib(0)) + fib(0) : 1 + 0

Give it a try. The goal is to get you to think in new ways and to understand the final result.

Solution

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:

        result = fib(n-1) + fib(n-2)
        return result

print(fib(3))

print(fib(4))
```

2nd Python Problem for you to Solve

Previously we generated 1 number in the Fibonacci sequence. This time ask the user to define how many numbers they want and display them

Remember the formula for calculating the Fibonacci sequence is
Fn = Fn-1 + Fn-2
Where F0 = 0 and F1 = 1

Here is sample output you should aim for

How many Fibonacci values should be found : 30
1
1
2
3
5
All Done

You'll use the same function above with a while loop for output.

Solution

```
def fib(num):
    if num == 0:
        return 0
    elif num == 1:
        return 1
    else:
        result = fib(num - 1) + fib(num - 2)
        return result

numFibValues = int(input("How many Fibonacci values should be found : "))

i = 1

# While i is less then the number of values requested
# continue to find more
while i < numFibValues:

    # Call the fib()
    fibValue = fib(i)

    print(fibValue)

    i += 1

print("All Done")
```

I hope you enjoyed that tutorial. In the next video I'll teach you how to read and write files and I'll cover tuples as an added bonus.

VIDEO 15 : File I/O & Tuples

This time we'll cover how to read and write files and we'll investigate what a tuple is.

Writing Text to a File

I'll jump directly into the code needed to write text to a file.

CODE

```
# The os module provides methods for file processing
import os

# You can create or use an already created file with open

# If you use w (write) for mode then the file is
# overwritten.
# If you use a (append) you add to the end of the file

# Text is stored using unicode where numbers represent
# all possible characters

# We start the code with with which guarantees the file
# will be closed if the program crashes
with open("mydata.txt", mode="w", encoding="utf-8") as myFile:

    # You can write to the file with write
    # It doesn't add a newline
    myFile.write("Some random text\nMore random text\nAnd some more")
```

Reading Text from a File

Now we'll read text from a file and I'll show you how to perform some common directory procedures.

CODE

```
Import os

# Open the file for reading
# You don't have to provide a mode because it is
# read by default
with open("mydata.txt", encoding="utf-8") as myFile:

    # We can read data in a few ways
    # 1. read() reads everything into 1 string
    # 2. readline() reads everything including the first newline
    # 3. readlines() returns a list of every line which includes
    # each newline
```

```
    # Use read() to get everything at once
    print(myFile.read())

# Find out if the file is closed
print(myFile.closed)

# Get the file name
print(myFile.name)

# Get the access mode of the file
print(myFile.mode)

# Rename our file
os.rename("mydata.txt", "mydata2.txt")

# Delete a file
# os.remove("mydata.dat")

# Create a directory
# os.mkdir("mydir")

# Change directories
# os.chdir("mydir")

# Display current directory
print("Current Directory :", os.getcwd())

# Remove a directory, but 1st move back 1 directory
# os.chdir("..")
# os.rmdir("mydir")
```

Read One Line at a Time

You can read one line at a time with readline().

CODE

```
Import os

# Open the file
with open("mydata2.txt", encoding="utf-8") as myFile:

    lineNum = 1

    # We'll use a while loop that loops until the data
    # read is empty
    while True:
        line = myFile.readline()

        # line is empty so exit
        if not line:
            break

        print("Line", lineNum, " :", line, end="")
```

```
        lineNum += 1
```

Python Problem for you to Solve

For this problem I want you to cycle through each line of text and output the number of words and the average word length. Here is sample output.

```
Line 1
Number of Words : 3
Avg Word Length : 4.7
Line 2
Number of Words : 3
Avg Word Length : 4.7
```

We'll use the file we previously worked with.

Solution

```
Import os

with open("mydata2.txt", encoding="utf-8") as myFile:

    lineNum = 1

    while True:
        line = myFile.readline()

        # line is empty so exit
        if not line:
            break

        print("Line", lineNum)

        # Put the words in a list using the space as
        # the boundary between words
        wordList = line.split()

        # Get the number of words with len()
        print("Number of Words :", len(wordList))

        # Incremented for each character
        charCount = 0

        for word in wordList:
            for char in word:
                charCount += 1

        # Divide to find the answer
        avgNumChars = charCount/len(wordList)

        # Use format to limit to 2 decimals
        print("Avg Word Length : {:.2}".format(avgNumChars))
```

```
    lineNum += 1
```

Tuples

Now as a bonus I'll cover tuples. A Tuple is like a list, but their values can't be changed. Tuples are surrounded with parentheses instead of square brackets. Here is some sample code.

CODE

```
my_tuple = (1, 2, 3, 5, 8)

# Get a value with an index
print("1st Value :", my_tuple[0])

# Get a slice from the 1st index up to but not including
# the 3rd
print(my_tuple[0:3])

# Get the number of items in a Tuple
print("Tuple Length :", len(my_tuple))

# Join or concatenate tuples
more_fibs = my_tuple + (13, 21, 34)

# Check if a value is in a Tuple
print("34 in Tuple :", 34 in more_fibs)

# Iterate through a tuple
for i in more_fibs:
    print(i)

# Convert a List into a Tuple
a_list = [55, 89, 144]
a_tuple = tuple(a_list)

# Convert a Tuple into a List
a_list = list(a_tuple)

# Get max and minimum value
print("Min :", min(a_tuple))
print("Max :", max(a_tuple))
```

I hope you have enjoyed this tutorial. In the next part I'll cover Classes, Objects, Self, __init__, Getters, Setters, Properties, and then create 2 warriors that fight to the death!!!

Video 16 : Classes & Objects

In this tutorial I'll teach you how object oriented programming works with Python. Object Oriented Programming is the act of modeling real world objects in code.

Real world objects have attributes and capabilities. A person has the attributes of height, weight, name, etc. They also have the capability of being able to talk, walk, eat, etc.

With OOP we store the attributes in variables called fields. We then model capabilities using functions which are called methods.

A class is a blueprint we use to define an objects attributes (fields) and capabilities (methods). Here we will model a Dog object.

CODE

```
# Start by defining the objects name with
class Dog:
    # The init method is called to create an object
    # We give default values for the fields if none
    # are provided
    def __init__(self, name="", height=0, weight=0):
        # self allows an object to refer to itself
        # It is like how you refer to yourself with my
        # We will take the values passed in and assign
        # them to the new Dog objects fields (attributes)
        self.name = name
        self.height = height
        self.weight = weight

    # Define what happens when the Dog is asked to
    # demonstrate its capabilities
    def run(self):
        print("{} the dog runs".format(self.name))
    def eat(self):
        print("{} the dog eats".format(self.name))
    def bark(self):
        print("{} the dog barks".format(self.name))


def main():
    # Create a new Dog object
    spot = Dog("Spot", 66, 26)
    spot.bark()

main()
```

Getters & Setters

Getters and Setters are used to protect our objects from assigning bad fields or for providing improved output. If someone tries to assign “A” to weight block it. Maybe you want to provide lbs or kgs along with weight. With a getter you can do that.

Here is an example where we use both getters and setters while creating a Square object.

CODE

```
class Square:
    def __init__(self, height="0", width="0"):
        self.height = height
        self.width = width

    # This is the getter
    # @property defines that any call to height runs the code in the height method below it
    @property
    def height(self):
        print("Retrieving the height")

        # Put a __ before this private field
        return self.__height

    # This is the setter
    @height.setter
    def height(self, value):

        # We protect the height from receiving a bad value
        if value.isdigit():

            # Put a __ before this private field
            self.__height = value
        else:
            print("Please only enter numbers for height")

    # This is the getter
    @property
    def width(self):
        print("Retrieving the width")
        return self.__width

    # This is the setter
    @width.setter
    def width(self, value):
        if value.isdigit():
            self.__width = value
        else:
            print("Please only enter numbers for width")

    def get_area(self):
        return int(self.__width) * int(self.__height)
```

```
def main():
    square = Square()

    height = input("Enter height : ")
    width = input("Enter width : ")

    square.height = height
  square.width = width

    print("Height :", square.height)
    print("Width :", square.width)

    print("The Area is :", square.get_area())
```

That's all for now. In the next video I'll create 2 warrior objects and have them fight to the death!

QUIZ

1. What function is called each time you create a new object? init()
2. What keyword is used to a class when an object wants to refer to itself? Self
3. How would the object spot call for the bark function to execute? spot.bark()
4. What keyword do we use to define any call to a field will execute a function? @property
5. What keyword would be used to define height as a setter? @height.setter

Video 17 : Simulating a Fight with Objects

In this tutorial we will have some fun with Object Oriented Programming, while simulating a fight between Thor and Loki. Sample output will look like this :

```
Thor attacks Loki and deals 9 damage
Loki is down to 10 health
Loki attacks Thor and deals 7 damage
Thor is down to 7 health
Thor attacks Loki and deals 19 damage
Loki is down to -9 health
Loki has Died and Thor is Victorious
Game Over
```

We will create classes for both a Warrior and a Battle class. The Warrior class will simulate both the attributes and capabilities of a Warrior. The Battle class will however simulate the actions that occur in a battle such as starting the fight and getting the results.

CODE

```
# We will create classes for both a Warrior and a Battle class
# The Warrior class will simulate both the attributes and capabilities of a Warrior
# The Battle class will however simulate the actions that occur in a battle such as starting the fight and getting the results

import random
import math

# Warriors will have names, health, and attack and block maximums
# They will have the capabilities to attack and block random amounts
class Warrior:
    def __init__(self, name="warrior", health=0, attk_max=0, block_max=0):
        self.name = name
        self.health = health
        self.attk_max = attk_max
        self.block_max = block_max

    def attack(self):
        # Randomly calculate the attack amount
        # random() returns a value from 0.0 to 1.0
        attk_amt = self.attk_max * (random.random() + .5)
        return attk_amt

    def block(self):
        # Randomly calculate how much of the attack was blocked
        block_amt = self.block_max * (random.random() + .5)
        return block_amt

# The Battle class will have the capability to loop until 1 Warrior dies
# The Warriors will each get a turn to attack each turn
```

```
class Battle:
    def start_fight(self, warrior1, warrior2):
        # Continue looping until a Warrior dies switching back and
        # forth as the Warriors attack each other
        while True:
            if self.get_attack_result(warrior1, warrior2) == "Game Over":
                print("Game Over")
                break

            if self.get_attack_result(warrior2, warrior1) == "Game Over":
                print("Game Over")
                break

    # A function will receive each Warrior that will attack the other
    # Have the attack and block amounts be integers to make the results clean
    # Output the results of the fight as it goes
    # If a Warrior dies return that result to end the looping in the
    # above function

    # Make this method static because we don't need to use self
    @staticmethod
    def get_attack_result(warriorA, warriorB):
        warrior_a_attk_amt = warriorA.attack()
        warrior_b_block_amt = warriorB.block()
        damage_2_warrior_b = math.ceil(warrior_a_attk_amt - warrior_b_block_amt)
        warriorB.health = warriorB.health - damage_2_warrior_b

        print("{} attacks {} and deals {} damage".format(warriorA.name, warriorB.name, dam-
age_2_warrior_b))
        print("{} is down to {} health".format(warriorB.name,
warriorB.health))

        if warriorB.health <= 0:
            print("{} has Died and {} is Victorious".format(warriorB.name, warriorA.name))
            return "Game Over"
        else:
            return "Fight Again"

def main():
    # Create 2 Warriors
    thor = Warrior("Thor", 50, 20, 10)
    loki = Warrior("Loki", 50, 20, 10)
    # Create Battle object
    battle = Battle()
    # Initiate Battle
    battle.start_fight(thor, loki)

main()
```

I hope you enjoyed that tutorial. It was fun to make! In the next video I'll cover inheritance, operator overloading and polymorphism.

Video 18 : Inheritance, Operator Overloading and Polymorphism

In this part of the tutorial I'll cover Inheritance, Operator Overloading, and Polymorphism.

When we create a class we can inherit all of the fields and methods from another class. This is called inheritance.

The class that inherits is called the subclass and the class we inherit from is the super class.

CODE

```
# This will be our super class
class Animal:
    def __init__(self, birth_type="Unknown", appearance="Unknown", blooded="Unknown"):
        self.__birth_type = birth_type
        self.__appearance = appearance
        self.__blooded = blooded

    # The getter method
    @property
    def birth_type(self):

    # When using getters and setters don't forget the __
        return self.__birth_type

    @birth_type.setter
    def birth_type(self, birth_type):
        self.__birth_type = birth_type

    @property
    def appearance(self):
        return self.__appearance

    @appearance.setter
    def appearance(self, appearance):
        self.__appearance = appearance

    @property
    def blooded(self):
        return self.__blooded

    @blooded.setter
    def blooded(self, blooded):
        self.__blooded = blooded

    # Can be used to cast our object as a string
    # type(self).__name__ returns the class name
    def __str__(self):
        return "A {} is {} it is {} it is " \
               "{}".format(type(self).__name__,
                                   self.birth_type,
                                   self.appearance,
```

```
                                    self.blooded)

# Create a Mammal class that inherits from Animal
# You can inherit from multiple classes by separating
# the classes with a comma in the parentheses
class Mammal(Animal):
    def __init__(self, birth_type="born alive",
                 appearance="hair or fur",
                 blooded="warm blooded",
                 nurse_young=True):

        # Call for the super class to initialize fields
        Animal.__init__(self, birth_type,
                        appearance,
                        blooded)

        self.__nurse_young = nurse_young

    # We can extend the subclasses
    @property
    def nurse_young(self):
        return self.__nurse_young

    @nurse_young.setter
    def appearance(self, nurse_young):
        self.__nurse_young = nurse_young

    # Overwrite __str__
    # You can use super() to refer to the superclass
    def __str__(self):
        return super().__str__() + " and it is {} they nurse " \
            "their young".format(self.nurse_young)


class Reptile(Animal):
    def __init__(self, birth_type="born in an egg or born alive",
                 appearance="dry scales",
                 blooded="cold blooded"):

        # Call for the super class to initialize fields
        Animal.__init__(self, birth_type,
                        appearance,
                        blooded)


def main():
    animal1 = Animal("born alive")

    print(animal1.birth_type)

    # Call __str__()
    print(animal1)
    print()
```

```
    mammal1 = Mammal()

    print(mammal1)

    print(mammal1.birth_type)
    print(mammal1.appearance)
    print(mammal1.blooded)
    print(mammal1.nurse_young)
    print()

    reptile1 = Reptile()

    print(reptile1.birth_type)
    print(reptile1.appearance)
    print(reptile1.blooded)

main()


    # Polymorphism in Python works differently from other
    # languages in that functions accept any object
    # and expect that object to provide the needed method

    # This isn't something to dwell on. Just know that
    # if you call on a method for an object that the
    # method just needs to exist for that object to work.
    # Polymorphism is a big deal in other languages that
    # are statically typed (type is defined at declaration)
    # but because Python is dynamically typed (type defined
    # when a value is assigned) it doesn't matter as much.

    def getBirthType(theObject):
        print("The {} is {}".format(type(theObject).__name__,
                                    theObject.birth_type))

    getBirthType(mammal1)
    getBirthType(reptile1)

main()
```

Video 19 : Magic Methods

Magic methods allow you to define how objects of the same object type can be compared. They also allow you to define what happens when mathematical operations are performed on your objects in awesome ways!

Magic methods are surrounded by double underscores. We can use magic methods to define how operators like +, -, *, /, ==, >, <, etc. will work with our custom objects.

Magic methods are used for operator overloading in Python. Here are the magic methods you can manipulate :

```
# __eq__ : Equal
# __ne__ : Not Equal
# __lt__ : Less Than
# __gt__ : Greater Than
# __le__ : Less Than or Equal
# __ge__ : Greater Than or Equal
# __add__ : Addition
# __sub__ : Subtraction
# __mul__ : Multiplication
# __div__ : Division
# __mod__ : Modulus
```

In this example I'll create a Time class. I'll then create custom methods that define how your Time objects are printed and added.

CODE

```
class Time:
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second

    # Magic method that defines the string format of the object
    def __str__(self):
        # :02d adds a leading zero to have a minimum of 2 digits
        return "{}:{:02d}:{:02d}".format(self.hour,self.minute, self.second)

    def __add__(self, other_time):
        new_time = Time()

        # ---------- PROBLEM ----------
        # How would you go about adding 2 times together?

        # Add the seconds and correct if sum is >= 60
        if (self.second + other_time.second) >= 60:
            self.minute += 1
            new_time.second = (self.second + other_time.second) - 60
        else:
            new_time.second = self.second + other_time.second
```

```
        # Add the minutes and correct if sum is >= 60
        if (self.minute + other_time.minute) >= 60:
            self.hour += 1
            new_time.minute = (self.minute + other_time.minute) - 60
        else:
            new_time.minute = self.minute + other_time.minute

        # Add the minutes and correct if sum is > 60
        if (self.hour + other_time.hour) > 24:
            new_time.hour = (self.hour + other_time.hour) - 24
        else:
            new_time.hour = self.hour + other_time.hour
        return new_time

def main():
    time1 = Time(1, 20, 30)
    print(time1)
    time2 = Time(24, 41, 30)
    print(time1 + time2)

    # For homework get the Time objects to work for the other
    # mathematical and comparison operators

main()
```

Video 20 : Static and Modules

In this tutorial I'll cover static methods, static variables, how to make your own modules, import and from.

Static Methods

Static methods allow access without the need to initialize a class. They should be used as utility methods, or when a method is needed, but it doesn't make sense for the real world object to be able to perform a task.

Here I'll create a Sum object that is only used to provide a utility get_sum() function.

CODE

```
class Sum:

    # You use the static method decorator to define that a
    # method is static
    @staticmethod
    def get_sum(*args):

        sum_1 = 0

        for i in args:
            sum_1 += i

        return sum_1

def main():

    # Call a static method by proceeding it with its class
    # name
    print("Sum :", Sum.get_sum(1,2,3,4,5))


main()
```

Static Variables

Fields declared in a class, but outside of any method are static variables. Their value is shared by every object of that class.

CODE

```
class Dog:

    # This is a static variable
    num_of_dogs = 0

    def __init__(self, name="Unknown"):
        self.name = name
```

```
        # You reference the static variable by proceeding
        # it with the class name
        Dog.num_of_dogs += 1

    @staticmethod
    def get_num_of_dogs():
        print("There are currently {} dogs".format(Dog.num_of_dogs))

def main():

    spot = Dog("Spot")

    doug = Dog("Doug")

    spot.get_num_of_dogs()


main()
```

Modules

Your Python programs will contain a main program that includes your main function. Then you will create many modules in separate files. Modules also end with .py just like any other Python file.

CODE

```
# ————— sum.py —————
def get_sum(*args):
    sum_1 = 0

    for i in args:
        sum_1 += i

    return sum_1

# ————— End of sum.py —————

# You can import by listing the file name minus the py
import sum

# Get access to functions by proceeding with the file
# name and then the function you want
print("Sum :", sum.get_sum(1,2,3,4,5))
```

FROM

You can use from to copy specific functions from a module. You can use from sum import * to import all functions. You can also import multiple functions by listing them after import separated by commas.

CODE

```
from sum import get_sum

# You don't have to reference the module name now
print("Sum :", get_sum(1,2,3,4,5))
```

Video 21 : Custom Exceptions

I cover exception handling in a previous video, but this time I'll show you how to create custom exceptions. I'll also cover how to use finally and else with exceptions.

Exceptions are triggered either when an error occurs or when you want them to.

We know exceptions are used to handle errors, execute specific code when code generates something out of the ordinary, or to always execute code when something happens (close a file that was opened),

When an error occurs you stop executing code and jump to execute other code that responds to that error.

In this example let's handle an IndexError exception that is triggered when you try to access an index in a list that doesn't exist.

CODE

```
# Surround a potential exception with try
try:
    a_list = [1, 2, 3]

    print(a_list[3])

# Catch the exception with except followed by the
# exception you want to catch

# You can catch multiple exceptions by separating them
# with commas inside parentheses
# except (IndexError, NameError):
except IndexError:
    print("Sorry that index doesn't exist")

# If the exception wasn't caught above this will
# catch all others
except:
    print("An unknown error occurred")
```

Custom Exceptions

Now let's create a custom exception. Lets trigger an exception if the user enters a name that contains a number. Although you won't commonly create your own exceptions
this is how you do it.

CODE

```
# Create a class that inherits from Exception
class DogNameError(Exception):

    def __init__(self, *args, **kwargs):
        Exception.__init__(self, *args, **kwargs)
```

```
try:
    dog_name = input("What is your dogs name : ")

    if any(char.isdigit() for char in dog_name):

        # Raise your own exception
        # You can raise the built in exceptions as well
        raise DogNameError

except DogNameError:
    print("Your dogs name can't contain a number")
```

Finally & Else

Finally is used when you always want certain code to execute whether an exception is raised or not. Else is only executed if no exception was raised.

CODE

```
num1, num2 = input("Enter to values to divide : ").split()

try:
    quotient = int(num1) / int(num2)
    print("{} / {} = {}".format(num1, num2, quotient))

except ZeroDivisionError:
    print("You can't divide by zero")

# else is only executed if no exception was raised
else:
    print("You didn't raise an exception")

finally:
    print("I execute no matter what")
```

Problem for you to Solve

1. Create a file named mydata2.txt and put data in it
2. Using what you learned in part 8 and Google to find out how to open a file without with try to open the file in a try block
3. Catch the FileNotFoundError exception
4. In else print the file contents
5. In finally close the file
6. Try to open the nonexistent file mydata3.txt and test to see if you caught the exception

SOLUTION

```
try:
    my_file = open("mydata2.txt", encoding="utf-8")

# We can use as to access data and methods in the
# exception class
except FileNotFoundError as ex:
```

```
    print("That file was not found")

    # Print out further data on the exception
    print(ex.args)

else:
    print("File :", my_file.read())
    my_file.close()

finally:
    print("Finished Working with File")
```

That's all for now. In the next video I'll show how to treat functions as objects, cover function annotations and provide a problem for you to solve.

Video 22 : Functions as Objects

In this video I'll explore how we can treat functions as objects which opens up a world of possibilities. We'll also explore function annotations. Then we'll present you with another problem for you to solve.

CODE

```
# Function multiplies a parameter by 2
def mult_by_2(num):
    return num * 2

# A function can be
# 1. Assigned to another name

times_two = mult_by_2

print("4 * 2 =", times_two(4))

# 2. Passed into other functions

def do_math(func, num):

    return func(num)

print("8 * 2 =", do_math(mult_by_2, 8))

# 3. Returned from a function

def get_func_mult_by_num(num):

    # Create a dynamic function that will receive a value
    # and then return that value times the value passed
    # into get_func_mult_by_num()
    def mult_by(value):
        return num * value

    return mult_by

generated_func = get_func_mult_by_num(5)

print("5 * 10 =", generated_func(10))

# 4. Embedded in a data structure

list_of_funcs = [times_two, generated_func]

print("5 * 9 =", list_of_funcs[1](9))
```

Python Problem for you to Solve

Now that we have explored new ways we can use functions let's try another problem. I want you to create a function that receives a list and a function. The function passed will return True

or False if a list value is odd. And then the surrounding function will return a list of odd numbers.

Solution

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

def change_list(list, func):

    odd_list = []

    for i in list:
        if func(i):
            odd_list.append(i)

    return odd_list

a_list = range(1, 21)

print(change_list(a_list, is_it_odd))
```

Function Annotations

It is possible to define the data types of attributes and the returned value with annotations, but they have no impact on how the function operates, but instead are for documentation.

```
def random_func(name: str, age: int, weight: float) -> str:
    print("Name :", name)
    print("Age :", age)
    print("Weight :", weight)

    return "{} is {} years old and weighs {}".format(name, age, weight)

print(random_func("Derek", 41, 165.5))

# You don't get an error if you pass bad data
print(random_func(89, "Derek", "Turtle"))

# You can print the annotations
print(random_func.__annotations__)
```

That's it for this video. In the next video we'll cover Anonymous functions, lambda, map, filter, reduce and 2 new problems.

Video 23 : Anonymous Functions and More

This video will be fun! We will cover Anonymous functions, lambda, map, filter, reduce and 2 new problems. Lambda can be used to create anonymous functions. Lambda is like def, but rather than assign the function to a name it just returns it. Because there is no name that is why they are called anonymous functions. You can however assign a lambda function to a name.

This is their format
lambda arg1, arg2,... : expression using the args

Lambdas are used when you need a small function, but don't want to junk up your code with temporary function names that may cause conflicts. Let's look at some examples.

CODE

```
# Add values
sum_1 = lambda x, y : x + y

print("Sum :", sum_1(4, 5))

# Use a ternary operator to see if someone can vote
can_vote = lambda age: True if age >= 18 else False

print("Can Vote :", can_vote(16))

# Create a list of functions
power_list = [lambda x: x ** 2,
              lambda x: x ** 3,
              lambda x: x ** 4]

# Run each function on a value
for func in power_list:
    print(func(4))


# You can also store lambdas in dictionaires

attack = {'quick': (lambda: print("Quick Attack")),
          'power': (lambda: print("Power Attack")),
          'miss': (lambda: print("The Attack Missed"))}

attack['quick']()

# You could get a random dictionary as well for say our
# previous warrior objects
import random

# keys() returns an iterable so we convert it into a list
# choice() picks a random value from that list
attack_key = random.choice(list(attack.keys()))

attack[attack_key]()
```

Python Problem for you to Solve

Now that we have seen examples, let's try to solve a problem using what you've learned. Create a random list filled with the characters H and T for heads and tails. Output the number of Hs and Ts

Example Output
Heads : 46
Tails : 54

Solution

```
# Create the list
flip_list = []

# Populate the list with 100 Hs and Ts
# Trick : random.choice() returns a random value from the list
for i in range(1, 101):
    flip_list += random.choice(['H', 'T'])

# Output results
print("Heads : ", flip_list.count('H'))
print("Tails : ", flip_list.count('T'))
```

Map

Map allows us to execute a function on each item in a list. Let's look at why that is powerful.

CODE

```
# Generate a list from 1 to 10
one_to_10 = range(1, 11)

# The function to pass into map
def dbl_num(num):
    return num * 2

# Pass in the function and the list to generate a new list
print(list(map(dbl_num, one_to_10)))

# You could do the same thing with a lambda
print(list(map((lambda x: x * 3), one_to_10)))

# You can perform calculations against multiple lists
a_list = list(map((lambda x, y: x + y), [1, 2, 3], [1, 2, 3]))
print(a_list)
```

Filter

While map executes functions on a list, filter selects items from a list based on a function.

CODE

```
# Print out the even values from a list
print(list(filter((lambda x: x % 2 == 0), range(1, 11))))
```

Python Problem for you to Solve

Time for another problem that will test what you have learned. Find the multiples of 9 from a random 100 value list with values between 1 and 1000.

Solution

```
# Generate a random list with randint between 1 and 1000
# Use range to generate 100 values
rand_list = list(random.randint(1, 1001) for i in range(100))

# Use modulus to find multiples of 9 by passing the random
# list to filter
print(list(filter((lambda x: x % 9 == 0), rand_list)))
```

Reduce

Reduce is similar to map and filter, but it instead receives a list and returns a single result.

CODE

```
# You must import reduce
from functools import reduce

# Add up the values in a list
print(reduce((lambda x, y: x + y), range(1, 6)))
```

I hope you enjoyed this video. In the next video I'll cover iterables and show how you can add iterable behaviors to your classes using magic methods.

Video 24 : Iterables

In this video I'll cover iterables and show how you can add iterable behaviors to your classes using magic methods.

An iterable is a stored sequence of values (list) or, as we will see when we cover generators, an object that produces one value at a time.

Iterables differ from iterators in that an iterable is an object with an __iter__ method which returns an iterator. An iterator is an object with a __next__ method which retrieves the next value from sequence of values. Let's see an example.

CODE

```
# Define a string and convert it into an iterator
samp_str = iter("Sample")

print("Char :", next(samp_str))
print("Char :", next(samp_str))
```

Custom Iterable

Now I'll show how you can add iterator behavior to your custom classes.

CODE

```
class Alphabet:

    def __init__(self):
        self.letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        self.index = -1

    def __iter__(self):
        return self

    def __next__(self):
        if self.index >= len(self.letters) - 1:
            raise StopIteration
        self.index += 1
        return self.letters[self.index]

alpha = Alphabet()

for letter in alpha:
    print(letter)

# Iterate through a dictionary because it is an iterable
derek = {"fName": "Derek", "lName": "Banas"}

for key in derek:
    print(key, derek[key])
```

Python Problem for you to Solve

It's time for another problem. Create a class that returns values from the Fibonacci sequence each time next is called.

Sample Output
Fib : 1
Fib : 2
Fib : 3
Fib : 5

Solution

```
class FibGenerator:

    def __init__(self):
        self.first = 0
        self.second = 1

    def __iter__(self):
        return self

    def __next__(self):
        fib_num = self.first + self.second
        self.first = self.second
        self.second = fib_num
        return fib_num

fib_seq = FibGenerator()

for i in range(10):
    print("Fib :", next(fib_seq))
```

That's all for now. In the next video I'll cover list comprehensions, generator functions, and generator expressions.

Video 25 : List Comprehensions / Generators

In this video I'll cover list comprehensions, generator functions, and generator expressions. A list comprehension executes an expression against an iterable, which I covered in the last video.

Note: While they are super powerful, try not to make list comprehensions that are hard to figure out for others. Here are some examples of what you can do with them.

CODE

```
# To multiply 2 times every value with a map we'd do
print(list(map((lambda x: x * 2), range(1, 11))))

# With a list comprehension we'd do
# Note that a list comprehension is surrounded by []
# because it returns a list
print([2 * x for x in range(1, 11)])

# To construct a list of odds using filter we'd
print(list(filter((lambda x: x % 2 != 0), range(1, 11))))

# To do the same with a list comprehension
print([x for x in range(1, 11) if x % 2 != 0])

# A list comprehension can act as map and filter
# on one line
# Generate a list of 50 values and take them to the power
# of 2 and return all that are multiples of 8

print([i ** 2 for i in range(50) if i % 8 == 0])

# You can have multiple for loops as well
# Multiply all values in one list times all values in
# another
print([x * y for x in range(1, 3) for y in range(11, 16)])

# You can put list comprehensions in list comprehensions
# Generate a list of 10 values, multiply them by 2 and
# return multiples of 8
print([x for x in [i * 2 for i in range(10)] if x % 8 == 0])
```

Python Problem for you to Solve

Generate a list of 50 random values between 1 and 1000 and return those that are multiples of 9. You'll have to use a list comprehension in a list comprehension. This is a hard one!

CODE

```
import random

print([x for x in [random.randint(1, 1001) for i in range(50)] if x % 9 == 0])
```

```
# List comprehensions also make it easy to work with
# multidimensional lists

multi_list = [[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]]

print([col[1] for col in multi_list])

# Get the diagonal by incrementing 0, 0 -> 1, 1 -> 2, 2
print([multi_list[i][i] for i in range(len(multi_list))])
```

Generator Functions

A generator function returns a result generator when called. They can be suspended and resumed during execution of your program to create results over time rather then all at once.

We use generators when we want to big result set, but we don't want to slow down the program by creating it all at one time.

Here I'll create a generator that calculates primes and returns the next prime on command.

CODE

```
def is_prime(num):
    # This for loop cycles through primes from 2 to
    # the value to check
    for i in range(2, num):

        # If any division has no remainder we know it
        # isn't a prime number
        if (num % i) == 0:
            return False
    return True

# This is the generator
def gen_primes(max_number):

    # This for loop cycles through primes from 2 to
    # the maximum value requested
    for num1 in range(2, max_number):

        if is_prime(num1):

            # yield is what makes this a generator
            # When called by next it will return the
            # next result
            yield num1

# Create a reference to the generator
prime = gen_primes(50)

# Call next for each result
```

```
print("Prime :", next(prime))
print("Prime :", next(prime))
print("Prime :", next(prime))
```

Generator Expressions

Generator expressions look just like list comprehensions but they return results one at a time. The are surrounded by parentheses instead of [ ]

CODE

```
double = (x * 2 for x in range(10))

print("Double :", next(double))
print("Double :", next(double))

# You can iterate through all results as well
for num in double:
    print(num)
```

That's it for now. The next video will be a big one and we will focus on threads. We'll learn about sleep(), strftime(), the Threading Module, Creating Threads, activeCount(), enumerate(), Subclassing Threads, run(), start(), is_alive(), getName(), setName(), join(), Synchronizing Threads, acquire(), release(), Lock() and more.

Video 26 : Threads

This video will focus 100% on threads. We'll learn about sleep(), strftime(), the Threading Module, Creating Threads, activeCount(), enumerate(), Subclassing Threads, run(), start(), is_alive(), getName(), setName(), join(), Synchronizing Threads, acquire(), release(), Lock() and more.

When you use threads it is like you are running multiple programs at once.

Threads actually take turns executing. While one executes the other sleeps until it is its turn to execute. Here is an example.

CODE

```
import threading
import time
import random

def execute_thread(i):

    # strftime or string formatted time allows you to
    # define how the time is displayed.
    # You could include the date with
    # strftime("%Y-%m-%d %H:%M:%S", gmtime())

    # Print when the thread went to sleep
    print("Thread {} sleeps at {}".format(i,
                time.strftime("%H:%M:%S", time.gmtime())))

    # Generate a random sleep period of between 1 and
    # 5 seconds
    rand_sleep_time = random.randint(1, 5)

    # Pauses execution of code in this function for
    # a few seconds
    time.sleep(rand_sleep_time)

    # Print out info after the sleep time
    print("Thread {} stops sleeping at {}".format(i,
                time.strftime("%H:%M:%S", time.gmtime())))

for i in range(10):

    # Each time through the loop a Thread object is created
    # You pass it the function to execute and any
    # arguments to pass to that method
    # The arguments passed must be a sequence which
    # is why we need the comma with 1 argument
    thread = threading.Thread(target=execute_thread, args=(i,))
    thread.start()

    # Display active threads
    # The extra 1 is this for loop executing in the main
    # thread
```

```
    print("Active Threads :", threading.activeCount())

    # Returns a list of all active thread objects
    print("Thread Objects :", threading.enumerate())
```

Subclassing Threads

You can subclass the Thread object and then define what happens each time a new thread is executed or run.

CODE

```
class CustThread(threading.Thread):

    def __init__(self, name):
        threading.Thread.__init__(self)

        self.name = name

    def run(self):

        get_time(self.name)

        print("Thread", self.name, "Execution Ends")

def get_time(name):
    print("Thread {} sleeps at {}".format(name,
                time.strftime("%H:%M:%S", time.gmtime())))

    randSleepTime = random.randint(1, 5)

    time.sleep(randSleepTime)

# Create thread objects
thread1 = CustThread("1")
thread2 = CustThread("2")

# Start thread execution of run()
thread1.start()
thread2.start()

# Check if thread is alive
print("Thread 1 Alive :", thread1.is_alive())
print("Thread 2 Alive :", thread2.is_alive())

# Get thread name
# You can change it with setName()
print("Thread 1 Name :", thread1.getName())
print("Thread 2 Name :", thread2.getName())

# Wait for threads to exit
thread1.join()
thread2.join()
```

```
print("Execution Ends")
```

Synchronizing Threads

You can lock other threads from executing. If we try to model a bank account we have to make sure the account is locked down during a transaction so that if more then 1 person tries to withdrawal money at once we don't give out more money than is in the account.

```
class BankAccount (threading.Thread):

    acct_balance = 100

    def __init__(self, name, money_request):
        threading.Thread.__init__(self)
        self.name = name
        self.money_request = money_request

    def run(self):
        # Get lock to keep other threads from accessing the account
        threadLock.acquire()

        # Call the static method
        BankAccount.get_money(self)

        # Release lock so other thread can access the account
        threadLock.release()

    @staticmethod
    def get_money(customer):
        print("{} tries to withdrawal ${} at {}".format(customer.name,
              customer.money_request,
              time.strftime("%H:%M:%S", time.gmtime())))

        if BankAccount.acct_balance - customer.money_request > 0:
            BankAccount.acct_balance -= customer.money_request
            print("New account balance is : ${}".format(BankAccount.acct_balance))
        else:
            print("Not enough money in the account")
            print("Current balance : ${}".format(BankAccount.acct_balance))

        time.sleep(3)

# Create a lock to be used by threads
threadLock = threading.Lock()

# Create new threads
doug = BankAccount("Doug", 1)
paul = BankAccount("Paul", 100)
sally = BankAccount("Sally", 50)

# Start new Threads
doug.start()
```

```
paul.start()
sally.start()

# Have threads wait for previous threads to terminate
doug.join()
paul.join()
sally.join()

print("Execution Ends")
```

That’s all for now. In the next video I’ll start my multipart tutorial on Regular Expressions.

Video 27 : Regular Expressions

Regular expressions allow you to locate and change strings in very powerful ways. They work in almost exactly the same way in every programming language as well.

Regular Expressions (Regex) are used to :

1. Search for a specific string in a large amount of data
2. Verify that a string has the proper format (Email, Phone #)
3. Find a string and replace it with another string
4. Format data into the proper form for importing for example

First we'll search for an exact string match.

CODE

```
# import the Regex module
import re

# Search for ape in the string
if re.search("ape", "The ape was at the apex"):
    print("There is an ape")
```

Get All Matches

findall() returns a list of matches and . is used to match any 1 character or space. Finditer can be used to return an iterator of matches.

CODE

```
all_apes = re.findall("ape.", "The ape was at the apex")

for i in all_apes:
    print(i)

# finditer returns an iterator of matching objects
# You can use span to get the location

the_str = "The ape was at the apex"

for i in re.finditer("ape.", the_str):

    # Span returns a tuple
    loc_tuple = i.span()

    print(loc_tuple)

    # Slice the match out using the tuple values
    print(the_str[loc_tuple[0]:loc_tuple[1]])
```

Match 1 of Several Letters

Square brackets will match any one of the characters between the brackets not including upper and lowercase varieties unless they are listed. We can also define characters in a range and define that we want to match anything except a defined number of characters.

CODE

```
animal_str = "Cat rat mat fat pat"

all_animals = re.findall("[crmfp]at", animal_str)
for i in all_animals:
    print(i)

print()

# We can also allow for characters in a range
# Remember to include upper and lowercase letters
some_animals = re.findall("[c-mC-M]at", animal_str)
for i in some_animals:
    print(i)

print()

# Use ^ to denote any character but whatever characters are
# between the brackets
some_animals = re.findall("[^Cr]at", animal_str)
for i in some_animals:
    print(i)

print()
```

Replace All Matches

You can replace items and define pattern objects.

CODE

```
# Replace matching items in a string

owl_food = "rat cat mat pat"

# You can compile a regex into pattern objects which
# provide additional methods
regex = re.compile("[cr]at")

# sub() replaces items that match the regex in the string
# with the 1st attribute string passed to sub
owl_food = regex.sub("owl", owl_food)

print(owl_food)
```

Solving Backslash Problems

Regex use the backslash to designate special characters and Python does the same inside strings which causes issues.

CODE

```
# Let's try to get "\\stuff" out of a string
rand_str = "Here is \\stuff"

# This won't find it
print("Find \\stuff : ", re.search("\\stuff", rand_str))

# This does, but we have to put in 4 slashes which is
# messy
print("Find \\stuff : ", re.search("\\\\stuff", rand_str))

# You can get around this by using raw strings which
# don't treat backslashes as special
print("Find \\stuff : ", re.search(r"\\stuff", rand_str))
```

That’s all for this video. In the next video I’ll provide much more on what you can do with regular expressions.

Video 28 : Regular Expressions 2

I continue teaching about Regular Expressions. We'll learn how to match any character, white-space, numbers, one or more items, and we'll learn how to tell if an email address is legitimate or not.

Matching Any Character

We saw that . matches any character, but what if we want to match a period. Backslash the period. You do the same with [,] and others.

CODE

```
rand_str = "F.B.I. I.R.S. CIA"
print("Matches :", len(re.findall(".\..\..", rand_str)))
```

Matching Whitespace

We can match many whitespace characters

CODE

```
rand_str = """This is a long
string that goes
on for many lines"""

print(rand_str)

# Remove newlines
regex = re.compile("\n")

rand_str = regex.sub(" ", rand_str)

print(rand_str)

# You can also match
# \b : Backspace
# \f : Form Feed
# \r : Carriage Return
# \t : Tab
# \v : Vertical Tab

# You may need to remove \r\n on Windows
```

Matching Any Single Number

CODE

```
# \d can be used instead of [0-9]
# \D is the same as [^0-9]

randStr = "12345"
```

```
print("Matches :", len(re.findall("\d", randStr)))
```

Matching Multiple Numbers

You can match multiple digits by following the \d with {numOfValues}

CODE

```
# Match 5 numbers only
if re.search("\d{5}", "12345"):
    print("It is a zip code")

# You can also match within a range
# Match values that are between 5 and 7 digits
num_str = "123 12345 123456 1234567"

print("Matches :", len(re.findall("\d{5,7}", num_str)))
```

Matching Any Single Letter or Number

CODE

```
# \w is the same as [a-zA-Z0-9_]
# \W is the same as [^a-zA-Z0-9_]

ph_num = "412-555-1212"

# Check if it is a phone number
if re.search("\w{3}-\w{3}-\w{4}", ph_num):
    print("It is a phone number")

# Check for valid first name between 2 and 20 characters
if re.search("\w{2,20}", "Ultraman"):
    print("It is a valid name")
```

Matching WhiteSpace

CODE

```
# \s is the same as [\f\n\r\t\v]
# \S is the same as [^\f\n\r\t\v]

# Check for valid first and last name with a space
if re.search("\w{2,20}\s\w{2,20}", "Toshio Muramatsu"):
    print("It is a valid full name")
```

Matching One or More

```
# + matches 1 or more characters

# Match a followed by 1 or more characters
```

```
print("Matches :", len(re.findall("a+", "a as ape bug")))
```

Python Problem for you to Solve

Create a Regex that matches email addresses from a list. Translate the following rules into a Regex.

1. 1 to 20 lowercase and uppercase letters, numbers, plus ._%+-
2. An @ symbol
3. 2 to 20 lowercase and uppercase letters, numbers, plus .-
4. A period
5. 2 to 3 lowercase and uppercase letters

Solution

```
emailList = "db@aol.com m@.com @apple.com db@.com"

print("Email Matches :", len(re.findall("[\w._%+-]{1,20}@[\w.-]{2,20}.[A-Za-z]{2,3}",
                                        emailList)))
```

In the next video I'll continue down the path of turning you into a Regex Expert!

Video 29 : Regular Expressions 3

In this video we will continue towards our path to become Regex Experts! Here is everything we have learned so far :

Regex Tricks We Have Learned

Did you find a match
if re.search("REGEX", my_string)

Get list of matches
print("Matches :", len(re.findall("REGEX", my_string)))

Get a pattern object
regex = re.compile("REGEX")

Substitute the match
my_string = regex.sub("substitution", my_string)

[ ] : Match what is in the brackets
[^] : Match anything not in the brackets
. : Match any 1 character or space
+ : Match 1 or more of what proceeds
\n : Newline
\d : Any 1 number
\D : Anything but a number
\w : Same as [a-zA-Z0-9_]
\W : Same as [^a-zA-Z0-9_]
\s : Same as [\f\n\r\t\v]
\S : Same as [^\f\n\r\t\v]
{5} : Match 5 of what proceeds the curly brackets
{5,7} : Match values that are between 5 and 7 in length

Matching Zero or One

```
rand_str = "cat cats"

regex = re.compile("[cat]+s?")

matches = re.findall(regex, rand_str)

# Match cat or cats
print("Matches :", len(matches))

for i in matches:
    print(i)
```

Matching Zero or More

```
rand_str = "doctor doctors doctor's"

# Match doctor doctors or doctor's
regex = re.compile("[doctor]+['s]*")
```

```
matches = re.findall(regex, rand_str)

print("Matches :", len(matches))

# You can do the same by setting an interval match
regex = re.compile("[doctor]+['s]{0,2}")

matches = re.findall(regex, rand_str)

print("Matches :", len(matches))

for i in matches:
    print(i)
```

Python Problem for you to Solve

On Windows newlines are some times \n and other times \r\n

Create a regex that will grab each of the lines in this string, print out the number of matches and each line.

Solution

```
long_str = '''Just some words
and some more\r
and more
'''

print("Matches :", len(re.findall(r"[\w\s]+[\r]?\n", long_str)))

matches = re.findall("[\w\s]+[\r]?\n", long_str)

for i in matches:
    print(i)
```

Greedy & Lazy Matching

```
rand_str = "<name>Life On Mars</name><name>Freaks and Geeks</name>"

# Let's try to grab everything between <name> tags
# Because * is greedy (It grabs the biggest match possible)
# we can't get what we want, which is each individual tag
# match
regex = re.compile(r"<name>.*</name>")

matches = re.findall(regex, rand_str)

print("Matches :", len(matches))

for i in matches:
    print(i)
```

```
# We want to grab the smallest match we use *?, +?, or
# {n,}? instead

regex = re.compile(r"<name>.*?</name>")

matches = re.findall(regex, rand_str)

print("Matches :", len(matches))

for i in matches:
    print(i)
```

Word Boundaries

We use word boundaries to define where our matches start and end

\b matches the start or end of a word

```
# If we want ape it will match ape and the beginning of apex
rand_str = "ape at the apex"

regex = re.compile(r"ape")

# If we use the word boundary
regex = re.compile(r"\bape\b")

matches = re.findall(regex, rand_str)

print("Matches :", len(matches))

for i in matches:
    print(i)
```

Feeling like a Regex Expert? More is coming in the next video!

Video 30 : Regular Expressions 4

We continue on our path towards becoming Regex Experts in this video. We'll cover String Boundaries, Subexpressions, and solve another problem.

String Boundaries

```
# ^ : Matches the beginning of a string if outside of
#     a [ ]
# $ : Matches the end of a string

# Grab everything from the start of the string to @
rand_str = "Match everything up to @"

regex = re.compile(r"^.*[^@]")

matches = re.findall(regex, rand_str)

print("Matches :", len(matches))

for i in matches:
    print(i)

# Grab everything from @ to the end of the line
rand_str = "@ Get this string"

regex = re.compile(r"[^@\s].*$")

matches = re.findall(regex, rand_str)

print("Matches :", len(matches))

for i in matches:
    print(i)

# Grab the 1st word of each line using the the multiline
# code which allows for the targeting of each line after
# a line break with ^
rand_str = '''Ape is big
Turtle is slow
Cheetah is fast'''

regex = re.compile(r"(?m)^.*?\s")

matches = re.findall(regex, rand_str)

print("Matches :", len(matches))

for i in matches:
    print(i)
```

Subexpressions

Subexpressions are parts of a larger expression. If you want to match for a large block, but only want to return part of it. To do that surround what you want with ( )

CODE

```
# Get just the number minus the area code
rand_str = "My number is 412-555-1212"

regex = re.compile(r"412-(.*)")

matches = re.findall(regex, rand_str)

print("Matches :", len(matches))

for i in matches:
    print(i)
```

Python Problem for you to Solve

Get just the numbers minus the area codes from this string to solve this problem.

```
rand_str = "412-555-1212 412-555-1213 412-555-1214"
```

Solution

```
rand_str = "412-555-1212 412-555-1213 412-555-1214”
regex = re.compile(r"412-(.{8})")

matches = re.findall(regex, rand_str)

print("Matches :", len(matches))

for i in matches:
    print(i)
```

Multiple Subexpressions

```
# You can have multiple subexpressions as well
# Get both numbers that follow 412 separately
rand_str = "My number is 412-555-1212"

regex = re.compile(r"412-(.*)-(.*)")

matches = re.findall(regex, rand_str)

print("Matches :", len(matches))

print(matches[0][0])
print(matches[0][1])
```

You are getting very good at using Regular Expressions at this point. Now we take it to the next level. Next time I’ll cover Back References, Back Reference Substitutions, Look Ahead, Look Behind, and Negative Look Ahead & Behind.

Video 31 : Regular Expressions 5

In this video we cover advanced Regular Expressions techniques. I'll cover Back References, Back Reference Substitutions, Look Ahead, Look Behind, and Negative Look Ahead & Behind.

Everything We Have Learned

```
# Did you find a match
# if re.search("REGEX", my_string)

# Get list of matches
# print("Matches :", len(re.findall("REGEX", my_string)))

# Get a pattern object
# regex = re.compile("REGEX")

# Substitute the match
# my_string = regex.sub("substitution", my_string)

# [ ]   : Match what is in the brackets
# [^ ]  : Match anything not in the brackets
# ( )   : Return surrounded submatch
# .     : Match any 1 character or space
# +     : Match 1 or more of what proceeds
# ?     : Match 0 or 1
# *     : Match 0 or More
# *?    : Lazy match the smallest match
# \b    : Word boundary
# ^     : Beginning of String
# $     : End of String
# \n    : Newline
# \d    : Any 1 number
# \D    : Anything but a number
# \w    : Same as [a-zA-Z0-9_]
# \W    : Same as [^a-zA-Z0-9_]
# \s    : Same as [\f\n\r\t\v]
# \S    : Same as [^\f\n\r\t\v]
# {5}   : Match 5 of what proceeds the curly brackets
# {5,7} : Match values that are between 5 and 7 in length
# ($m)  : Allow ^ on multiline string
```

Back References

A back reference allows you to to reuse the expression that proceeds it

```
# Grab a double word
rand_str = "The cat cat fell out the window"

# Match a word boundary, 1 or more characters followed
# by a space if it is then followed by the same
# match that is surrounded by the parentheses
regex = re.compile(r"(\b\w+)\s+\1")
```

```
matches = re.findall(regex, rand_str)

print("Matches :", len(matches))

for i in matches:
    print(i)
```

Back Reference Substitutions

```
# Replace the bold tags in the link with no tags
rand_str = "<a href='#'><b>The Link</b></a>"

# Regex matches bold tags and grabs the text between
# them to be used by the back reference
regex = re.compile(r"<b>(.*?)</b>")

# Replace the tags with just the text between them
rand_str = re.sub(regex, r"\1", rand_str)

print(rand_str)
```

Another Back Reference Substitution

```
# Receive this string
rand_str = "412-555-1212"

# Match the phone number using multiple subexpressions
regex = re.compile(r"([\d]{3})-([\d]{3}-[\d]{4})")

# Output (412)555-1212
rand_str = re.sub(regex, r"(\1)\2", rand_str)

print(rand_str)
```

Python Problem for you to Solve

To solve this problem I want you to receive a string like this :

```
rand_str = "https://www.youtube.com http://www.google.com"
```

And, then Grab the URL and provide the following output using a back reference substitution :

```
<a href='https://www.youtube.com'>www.youtube.com</a>
<a href='https://www.google.com'>www.google.com</a>
```

Solution

```
regex = re.compile(r"(https?://([\w.]+))")

rand_str = re.sub(regex, r"<a href='\1'>\2</a>\n", rand_str)

print(rand_str)
```

Look Ahead

A look ahead defines a pattern to match but not return. You define the expression to look for but not return like this (?=expression)

```
rand_str = "One two three four"

# Grab all letters and numbers of 1 or more separated
# by a word boundary but don't include it
regex = re.compile(r"\w+(?=\b)")

matches = re.findall(regex, rand_str)

for i in matches:
    print(i)
```

Look Behind

The look behind looks for what is before the text to return, but doesn't return it. It is defined like (?<=expression)

```
rand_str = "1. Bread 2. Apples 3. Lettuce"

# Find the number, period and space, but only return
# the 1 or more letters or numbers that follow
regex = re.compile(r"(?<=\d.\s)\w+")

matches = re.findall(regex, rand_str)

for i in matches:
    print(i)
```

Look Ahead & Behind

```
rand_str = "<h1>I'm Important</h1> <h1>So am I</h1>"

# Use the look behind, get 1 or more of anything,
# and use the look ahead
regex = re.compile(r"(?<=<h1>).+?(?=</h1>)")

matches = re.findall(regex, rand_str)

for i in matches:
    print(i)
```

Negative Look Ahead & Behind

These are used to look for text that doesn't match the pattern

(?!expression) : Negative Look Ahead
(?<!expression) : Negative Look Behind

```
rand_str = "8 Apples $3, 1 Bread $1, 1 Cereal $4"
```

```
# Grab the total number of grocery items by ignoring the $
regex = re.compile(r"(?<!\$)\d+")

matches = re.findall(regex, rand_str)

print(len(matches))

# Convert from a string list to an int list
matches = [int(i) for i in matches]

from functools import reduce

# Sum the items in the list with reduce
print("Total Items {}".format(reduce((lambda x, y: x + y), matches)))
```

That's it for now. In the next video, I finish my coverage of Regular Expressions. We'll look at Or, Group, Named Groups, More Match Object Functions and then we'll solve some problems.

Video 36 : Regular Expressions 6

In this video I finish my Regular Expressions coverage. We'll look at Or, Group, Named Groups, More Match Object Functions and then we'll solve some problems.

Overview of What We've Learned About Regex

```
# [ ]   : Match what is in the brackets
# [^ ]  : Match anything not in the brackets
# ( )   : Return surrounded submatch
# .     : Match any 1 character or space
# +     : Match 1 or more of what proceeds
# ?     : Match 0 or 1
# *     : Match 0 or More
# *?    : Lazy match the smallest match
# \b    : Word boundary
# ^     : Beginning of String
# $     : End of String
# \n    : Newline
# \d    : Any 1 number
# \D    : Anything but a number
# \w    : Same as [a-zA-Z0-9_]
# \W    : Same as [^a-zA-Z0-9_]
# \s    : Same as [\f\n\r\t\v]
# \S    : Same as [^\f\n\r\t\v]
# {5}   : Match 5 of what proceeds the curly brackets
# {5,7} : Match values that are between 5 and 7 in length
# ($m)  : Allow ^ on multiline string

# Use a back reference to substitute what is between the
# bold tags and eliminate the bold tags
# re.sub(r"<b>(.*?)</b>", r"\1", randStr)

# Use a look ahead to find all characters of 1 or more
# with a word boundary, but don't return the word
# boundary
# re.findall(r"\w+(?=\b)", randStr)

# Use a look behind to find words starting with a number,
# period and space, but only return the word that follows
# re.findall(r"(?<=\d.\s)\w+", randStr)

# Use a negative look behind to only return numbers without
# a $ in front of them
# re.findall(r"(?<!\$)\d+", randStr)
```

Or Conditional

You can use | to define the matches you'll except

```
rand_str = "1. Dog 2. Cat 3. Turtle"

regex = re.compile(r"\d\.\s(Dog|Cat)")
```

```
matches = re.findall(regex, rand_str)

print(len(matches))

for i in matches:
    print(i)
```

Python Problem for you to Solve

Create a regex that will match for 5 digit zip codes or zip codes with 5 digits a dash and then 4 digits. Here is sample data :

```
rand_str = "12345 12345-1234 1234 12346-333"
```

Solution

```
rand_str = "12345 12345-1234 1234 12346-333”
regex = re.compile(r"(\d{5}-\d{4}|\d{5}\s)")

matches = re.findall(regex, rand_str)

print(len(matches))

for i in matches:
    print(i)
```

Group

We can use group to retrieve parts of regex matches

```
bd = input("Enter your birthday (mm-dd-yyyy) : ")

bd_regex = re.search(r"(\d{1,2})-(\d{1,2})-(\d{4})", bd)

print("You were born on", bd_regex.group())
print("Birth Month", bd_regex.group(1))
print("Birth Day", bd_regex.group(2))
print("Birth Year", bd_regex.group(3))
```

Match Object Functions

There are functions that provide more information on your matches

```
match = re.search(r"\d{2}", "The chicken weighed 13 lbs")

# Print the match
print("Match :", match.group())

# Print the start and ending index of the match
print("Span :", match.span())

# Print starting index of the match
```

```
print("Match :", match.start())

# Print the ending index of the match
print("Match :", match.end())
```

Named Groups

You can also assign names to matches.

```
rand_str = "December 21 1974"

regex = r"^(?P<month>\w+)\s(?P<day>\d+)\s(?P<year>\d+)"

matches = re.search(regex, rand_str)

print("Month :", matches.group('month'))
print("Day :", matches.group('day'))
print("Year :", matches.group('year'))
```

Python Problem for you to Solve

Find all of the following real email addresses in this sample data.

```
rand_str = "d+b@aol.com a_1@yahoo.co.uk A-100@m-b.INTERNATIONAL"
```

Solution

```
rand_str = "d+b@aol.com a_1@yahoo.co.uk A-100@m-b.INTERNATIONAL"
regex = re.compile(r"[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+")

matches = re.findall(regex, randStr)

print(len(matches))

for i in matches:
    print(i)
```

Python Problem for you to Solve

For your final Python / Regex problem I want you to match all of the following phone numbers and then print them.

```
rand_str = "14125551212 4125551212 (412)5551212 412 555 1212 412-555-1212
1-412-555-1212”
```

Solution

```
rand_str = "14125551212 4125551212 (412)5551212 412 555 1212 412-555-1212
1-412-555-1212”
regex = re.compile(r"((1?)(-| ?)(\()?(\d{3})(\)|-| |\)-|\) )?(\d{3})(-| )?(\d{4}|\d{4}))")

matches = re.findall(regex, randStr)
```

```
print(len(matches))

for i in matches:
    print(i[0].lstrip())
```

Thank you for taking this Regex journey with me. I hope that I was able to show you how powerful Regular Expressions can be, while at the same time making them easy to understand and grasp.

In the next part of my tutorial I'll show you how to work with databases using Python.

---------- SQLite3 Tutorial 1 ----------

SLIDE What is SQLite3

- **Relational Database**
- **Packaged with your Application**
- **Structured Query Language**
- **Queries**

SQLite3 is a relational database that is packaged inside your application. A relational database is a database that has strictly defined tables with specific rows and columns. You define, retrieve and alter that data using the Structured Query Language (SQL). You issue commands called queries using SQL.

SLIDE 2 Understanding Tables

Primary keys are unique numbers that identify unique piece of data stored in a table. Rows contain all the data that is specific to each entity. Columns represent the data that each entity has.

SLIDE 3 Primary & Foreign Keys

We join data stored in different tables by referring to the unique Primary Keys that identify each entity. When you store a primary key from another table that key is referred to as a foreign key.

```
-- Create a database

sqlite3 studentdb.db

-- Create the sex table because we want to define that it can only have either the value 'F' or
'M'
-- When you want to have a list of options other databases define you should use an enumer-
ated type which is a list of values. SQLite doesn't have enums so instead we will create a table
for sex

-- Marking data as a primary key means that an incrementing value will be added if no value is
provided

CREATE TABLE sex(
id TEXT PRIMARY KEY NOT NULL,
sex_type INTEGER);

-- You insert values into the sex table with insert

INSERT INTO sex(id, sex_type) VALUES ('M', 1);
INSERT INTO sex(id, sex_type) VALUES ('F', 2);

-- Display the values added
-- Format the output to show each record on a separate line with data aligned in columns
.mode column

-- Display the column names as headers
.headers on
```

```
-- Change our table column widths
.width 15 20

-- Display everything from the table sex
SELECT * FROM sex;

-- Display every value on its own line
.mode line
SELECT * FROM sex;

-- Display statement used to create the table
.schema sex

-- Create the student table
-- AUTOINCREMENT increments itself starting from 1 each time a new entity is entered
-- You define a foreign key by referencing the table and name for the id where the foreign key resides.

CREATE TABLE student(
f_name VARCHAR(23) NOT NULL,
l_name VARCHAR(23) NOT NULL,
sex CHARACTER(1) NOT NULL,
id INTEGER PRIMARY KEY AUTOINCREMENT,
FOREIGN KEY (sex) REFERENCES sex(id));
```

SLIDE 4 Data Types

- **Dynamic Type System**
- **Integer**
- **Real**
- **Text**
- **Blob**
- **Numeric**

SQLite uses a dynamic type system in which the data type is defined based on the type of data entered.

There are 5 data types being Integer, Real, Numeric, Text, Blob and Null. Integers are numbers without decimal places. Reals are numbers with decimal places. Text can store an unlimited number of characters. Blobs which stands for Binary Large OBject store any kind of data. A Null is anything without a value.

Numerics can contain values from any of the other types. It converts the data entered to different types as long as the data is stored and no data is lost. For example if you insert '123' it will convert it into an Integer. If a real type is entered as long as it is less than 15 decimal digits it will be stored as a Real and if larger it will be stored as a Text.

SLIDE 5 Data Type Conversion

- **Integer : INT, INTEGER, TINYINT, SMALLINT, …**
- **Real : REAL, DOUBLE, FLOAT, …**

- **Text : VARCHAR, CHARACTER, TEXT, …**
- **Blob : BLOB**
- **Numeric : DECIMAL, BOOL, DATE, …**

Any type containing INT is stored as an Integer. Any type containing Char or Text is a Text. Blobs are Blobs. Reals are any of the data types listed as well as Double Precision. Everything else except for Null is stored as a Numeric.

```
-- Create an enum that represents the type of test
CREATE TABLE test_type(
id INTEGER PRIMARY KEY NOT NULL,
type TEXT NOT NULL);

-- Insert values
INSERT INTO test_type(id, type) VALUES (1, 'Q');
INSERT INTO test_type(id, type) VALUES (2, 'T');

-- Create test table
CREATE TABLE test(
date DATE NOT NULL, -- DATE is stored as a NUMERIC type
test_type INT NOT NULL,
id INTEGER PRIMARY KEY AUTOINCREMENT,
FOREIGN KEY (test_type) REFERENCES test_type(id));

-- Table for test scores
-- I use a composite primary key here only to show you how to define them
-- They are typically not used because the more complicated an index is the more inefficient
the database will be

CREATE TABLE test_score(
student_id INTEGER NOT NULL,
test_id INTEGER NOT NULL,
score INTEGER NOT NULL,
FOREIGN KEY (test_id) REFERENCES test(id),
FOREIGN KEY (student_id) REFERENCES student(id),
PRIMARY KEY (test_id, student_id)); -- Composite Primary Key

-- Table for absences
CREATE TABLE absence(
student_id INTEGER NOT NULL,
date DATE NOT NULL,
PRIMARY KEY (student_id, date),
FOREIGN KEY (student_id) REFERENCES student(id));

-- Display all tables & views
.tables

-- Exit SQLite
.exit
```

That's all for now. In the next video I'll cover how to generate random data using Python to populate our databases. In the videos that follow that I'll show how to do just about anything with Python and SQLite3.

---------- SQLite3 Tutorial 2 ----------

```
import random

# I'll use these lists to generate random first and last names

f_names = ["Michael", "Christopher", "Joshua", "Matthew", "David",
          "Daniel", "Andrew", "Joseph", "Justin", "James",
          "Jessica", "Ashley", "Brittany", "Amanda", "Melissa",
          "Stephanie", "Jennifer", "Samantha", "Sarah", "Megan", "Lauren"]
l_names = ["Smith", "Johnson", "Williams", "Jones", "Brown",
          "Davis", "Miller", "Wilson", "Moore", "Taylor",
          "Anderson", "Thomas", "Jackson", "White", "Harris",
          "Martin", "Thompson", "Garcia", "Martinez", "Robinson"]


# Creates random student insert statements
def create_students(how_many):
    insert = "INSERT INTO student (f_name, l_name, sex) VALUES ('{}', '{}', '{}');"

    # randint returns a random value from the 1st value entered to the next
    for i in range(how_many):
        f_name = f_names[random.randint(0, len(f_names) - 1)]
        l_name = l_names[random.randint(0, len(l_names) - 1)]

        # choice returns a random value from the list provided
        sex = random.choice(['M', 'F'])
        print(f"INSERT INTO student (f_name, l_name, sex) VALUES ('{f_name}', '{l_name}',
'{sex}');")


create_students(10)

'''
TEST INSERTS

INSERT INTO test VALUES ('2018-10-1', 1, NULL);
INSERT INTO test VALUES ('2018-10-2', 2, NULL);
INSERT INTO test VALUES ('2018-10-4', 2, NULL);
INSERT INTO test VALUES (date('now'), 1, NULL);
'''


# Generate insert statements for test_scores with a random score
def create_test_scores(num_tests, num_studs):
    for i in range(1, num_tests+1):
        for j in range(1, num_studs+1):
            score = random.randrange(1, 25)
            print(f"INSERT INTO test_score VALUES ({j}, {i}, {score});")


create_test_scores(4, 10)
```

```
# Give students 1, 2, and 3 a -1 score because they were absent for that test
# Insert absences in the absence table
# INSERT INTO absence VALUES (1, '2018-10-1');
# INSERT INTO absence VALUES (2, '2018-10-1');
# INSERT INTO absence VALUES (3, '2018-10-1');

# In the next video I'll show you numerous select queries, joins and much more
```

---------- SQLite3 Tutorial 3 ----------

```
-- Show test results for all students for the quiz given on 2018-10-1
-- We need to pull this information from 2 tables this time

SELECT student_id, score, test_type, date
FROM test, test_score
WHERE date = '2018-10-1'
AND test.id = test_score.test_id;

-- Print out the students name with the scores
-- You have to match the student ids for tables test_score and student
-- That way they will only show the test score that corresponds with each
-- individual student

SELECT f_name, l_name, score, test_type, date
FROM test, test_score, student
WHERE date = '2018-10-1'
AND test.id = test_score.test_id
AND test_score.student_id = student.id;

-- List all students along with their number of absences
-- Since we are using an aggregate query here to group data we have to define
-- how we want the information to be grouped when it is displayed on the screen.
-- That is why we define id_number as the way to group information. It is saying
-- that we should calculate the number of absences for each id_number.

SELECT f_name || ' ' || l_name AS NAME,
COUNT(absence.date) AS ABSENCES
FROM student, absence
WHERE absence.student_id = student.id
GROUP BY id;

-- SQLite JOINS

-- Above we defined INNER JOINs by separating tables with a comma. You can also
-- define them with the word INNER JOIN

-- An INNER JOIN is the most common join. An INNER JOIN returns only those
-- records from tables that match. The JOIN CONDITION defines the results.

SELECT f_name || ' ' || l_name AS name, score, test_id
FROM test_score JOIN student
ON student_id = id;
```

```
-- To show all students with the number of absences even if they have none we
-- have to use a LEFT JOIN.

-- The LEFT JOIN says that we need a row for each piece of data listed on the
-- left of the join. Don't forget to change WHERE into ON

SELECT f_name || ' ' || l_name AS name,
COUNT(absence.date) AS absences
FROM student LEFT JOIN absence
ON absence.student_id = student.id
GROUP BY id;

-- A NATURAL INNER JOIN is similar to a LEFT JOIN in that it returns all columns
-- that match in both tables.

SELECT score, test_id
FROM student NATURAL JOIN test_score
WHERE student_id = id;

-- A CROSS INNER JOIN (Cartesian Join) combines all the records from 2 tables.
-- This can sometimes make a mess and should normally be avoided

SELECT score, test_id
FROM student CROSS JOIN test_score;

-- SQLites SELECT can also be used to perform numerous Arithmetic, Boolean,
-- Bitwise, Relational and other Operations

SELECT (1+2) / (6-3) * 10;

SELECT 15 % 10;

-- You can perform boolean operations in which 0 is false and any other number
-- is true

SELECT 1 AND 0, 1 OR 0, NOT 1;

-- Other Operators

SELECT 'Paul' IN ('Mike', 'Phil', 'Paul');

-- BETWEEN can be used to make comparisons as well

SELECT * FROM test_score;

SELECT * FROM test_score
WHERE score
BETWEEN 15 AND 20;

-- Generate minimum and maximum values from a result

SELECT min(id), max(id)
FROM student;
```

```
-- Returns the total number of changes made to the
-- database since it was last opened

SELECT total_changes();
```

---------- SQLite3 Tutorial 4 ----------

```
-- Applying Functions in SQLite

-- Find the Best and Worst Scores on all quizzes and tests

-- test_score : student_id, test_id, score
-- test : date, test_type, id
-- student : f_name, l_name, sex, id

SELECT test.date AS Date,
MIN(test_score.score) AS Worst,
MAX(test_score.score) AS Best
FROM test_score, test
WHERE test_score.test_id = test.id
GROUP BY test.date;

-- Print the average score on each test

SELECT test.date AS Date,
AVG(test_score.score) 'Avg Score'
FROM test_score, test
WHERE test_score.test_id = test.id
GROUP BY test.date;

-- List all students that had a test score over 20

SELECT f_name || ' ' || l_name AS Name, test_score.score AS Score
FROM test_score, student
WHERE test_score.score > 20 AND test_score.student_id = student.id
GROUP BY Name;

-- VIEWS IN SQLite --

-- A view is used to store a queries result. It is not part of the schema

CREATE VIEW ScoreOver20 AS
SELECT f_name || ' ' || l_name AS Name, test_score.score
FROM test_score, student
WHERE test_score.score > 20
AND test_score.student_id = student.id
GROUP BY Name;

drop view ScoreOver20; -- Delete the view

-- TRIGGERS in SQLite --

-- Triggers are operations that are automatically performed when a specific
```

```
-- event occurs

-- test : date, test_type, id
-- test_score : student_id, test_id, score
-- student : f_name, l_name, sex, id

-- Will Hold Data When a Student Has a Makeup Test

CREATE TABLE Log(
id INTEGER PRIMARY KEY,
test_id INTEGER NOT NULL,
date     DATE NOT NULL,
student_id INTEGER NOT NULL,
FOREIGN KEY (test_id) REFERENCES test_score (test_id),
FOREIGN KEY (student_id) REFERENCES test_score (student_id));

-- The Trigger that updates the Log when test_score is updated

CREATE TRIGGER test_score_update
AFTER UPDATE OF score ON test_score
BEGIN
INSERT INTO Log(test_id, date, student_id)
VALUES(new.test_id, date('now'), new.student_id);
-- Don't reference table instead use new
END;

select * from absence; -- Show all absences

UPDATE test_score
SET score=20
WHERE test_id=1 AND student_id=2;

-- LIKE can be used with % to match a series of characters and zero or more
-- characters there after

select l_name, f_name
from student
where l_name LIKE 'J%';

-- _ can be used to represent any 1 character or space
-- Match all 5 letter long last names

select l_name, f_name
from student
where l_name LIKE '_____';

-- ORDER BY allows you to define sorting either DESC or ASC
-- LIMIT allows you to limit your results
-- OFFSET will skip the first number or results

select l_name, f_name
from student
where f_name LIKE 'J%'
ORDER BY l_name ASC, f_name LIMIT 10 OFFSET 2;
```

```
-- Random SQLite Functions

SELECT random(); -- Generate random number

SELECT ABS(RANDOM() % 100); -- Random number between 0 and 100

SELECT LOWER(f_name),
UPPER(l_name)
FROM student;

SELECT LENGTH('Iron Man'); -- Returns the number of characters in a string

SELECT COUNT(*) FROM student; -- Number of rows in the table

SELECT date(); -- Return the current date

SELECT time(); -- Return the current time

SELECT datetime(); -- Return the current date and time

SELECT date('now', '-30 days'); -- Get the date 30 days ago

SELECT date('now', '-20 months'); -- Get the date 30 days ago

SELECT date('now', 'weekday 0'); -- Get the date of the next Sunday

SELECT time('now', '-1000 minutes');

SELECT time('now', '-1000 seconds');

SELECT strftime('%m-%d-%Y'); -- You can modify the date format

-- Find Thanksgiving day

SELECT date('now', 'start of year', '10 months', '21 days', 'weekday 4');
```

---------- SQLite3 & Python ----------

```
# Here I'll show you how to work with SQLite databases
# in Python

# A database makes it easy for you to organize your
# data for storage and fast searching

# I show how to install SQLite and use it in a previous video

# You need the SQLite module to use it
import sqlite3
import sys
import csv

# connect() will open an SQLite database, or if it
```

```
# doesn't exist it will create it
# The file appears in the same directory as this
# Python file
db_conn = sqlite3.connect('test.db')
print("Database Created")

# A cursor is used to traverse the records of a result
the_cursor = db_conn.cursor()

def print_db():
    # To retrieve data from a table use SELECT followed
    # by the items to retrieve and the table to
    # retrieve from
    try:
        result = the_cursor.execute("SELECT id, f_name, l_name, age, address, salary, hire_date
FROM employees")

        # You receive a list of lists that hold the result
        for row in result:
            print("id :", row[0])
            print("f_name :", row[1])
            print("l_name :", row[2])
            print("age :", row[3])
            print("address :", row[4])
            print("salary :", row[5])
            print("hire_date :", row[6])
    except sqlite3.OperationalError:
        print("The table doesn't exist")
    except:
        print("Couldn't retrieve data from database")


# execute() executes a SQL command
# We organize our data in tables by defining their
# name and the data type for the data

# We define the table name
# A primary key is a unique value that differentiates
# each row of data in our table
# The primary key will auto increment each time we
# add a new Employee
# If a piece of data is marked as NOT NULL, that means
# it must have a value to be valid

# NULL is NULL and stands in for no value
# INTEGER is an integer
# TEXT is a string of variable length
# REAL is a float
# BLOB is used to store binary data

# You can delete a table if it exists like this
# db_conn.execute("DROP TABLE IF EXISTS Employees")
# db_conn.commit()
try:
```

```
    db_conn.execute("CREATE TABLE employees(id INTEGER PRIMARY KEY AUTOINCRE-
MENT NOT NULL, f_name TEXT NOT NULL, l_name TEXT NOT NULL, age INT NOT NULL,
address TEXT, salary REAL, hire_date TEXT);")
    db_conn.commit()
    print("Table Created")
except sqlite3.OperationalError as e:
    print("Table couldn't be created :", str(e))

# To insert data into a table we use INSERT INTO
# followed by the table name and the item name
# and the data to assign to those items
db_conn.execute("INSERT INTO employees(f_name, l_name, age, address, salary, hire_date)
VALUES ('Derek', 'Banas', 43, '123 Main St', 500000, date('now'));")
db_conn.commit()
print("Employee Entered")

# Print out all the data in the database
print_db()

# You can update a value in a table by referencing
# something unique like the ID or anything else
# with the UPDATE command
try:
    db_conn.execute("UPDATE employees SET address = '121 Main St' WHERE ID = 1")
    db_conn.commit()
except sqlite3.OperationalError:
    print("Database couldn't be updated")

print_db()

# Delete matching data from the database by
# referencing the table name and something unique
try:
    db_conn.execute("DELETE FROM employees WHERE ID = 1")
    db_conn.commit()
except sqlite3.OperationalError:
    print("Data couldn't be deleted")

print_db()

# Undo the last commit()
db_conn.rollback()

print_db()

# You can add a new column to a table with ALTER
try:
    db_conn.execute("ALTER TABLE employees ADD COLUMN 'image' BLOB DEFAULT NULL")
    db_conn.commit()
except sqlite3.OperationalError:
    print("Table couldn't be altered")

# Retrieve table column names
the_cursor.execute("PRAGMA TABLE_INFO(employees)")
```

```
# fetchall() returns all remaining rows of a query result
# as a list
row_names = [nameTuple[1] for nameTuple in the_cursor.fetchall()]
print(row_names)

# Get the total number of rows
the_cursor.execute('SELECT COUNT(*) FROM employees')
num_of_rows = the_cursor.fetchall()
print("Total Rows :", num_of_rows[0][0])

# Get SQLite version
the_cursor.execute("SELECT SQLITE_VERSION()")

# fetchone() returns one result
print("SQLITE VERSION :", the_cursor.fetchone())

# Use the dictionary cursor to retrieve data in a dictionary
with db_conn:
    db_conn.row_factory = sqlite3.Row
    the_cursor = db_conn.cursor()
    the_cursor.execute("SELECT * FROM employees")
    rows = the_cursor.fetchall()
    for row in rows:
        print("{} {}".format(row["f_name"], row["l_name"]))

# Write data to File
with open('dump.sql', 'w') as f:
    # iterdump() returns an iterator to dump the database
    # in SQL format

    for line in db_conn.iterdump():
        f.write("%s\n" % line)

# Close the database connection
db_conn.close()
```

---------- Python Tkinter Tutorial ----------

```
'''
Tk is a GUI toolkit used for creating desktop
applications that work on Windows, MacOS and Linux
Tk provides tons of widgets (Buttons, Scrollbars, etc.)
that are used to create applications.

Tcl (Tool Command Language) is a programming
language used for developing web & desktop applications
'''

# Get the standard library for Tk
from tkinter import *
```

```
# Get the newest widget themes from Tk 8.5
from tkinter import ttk


def get_sum(*args):
    try:
        # Cast string to a float
        num_1_val = float(num_1.get())
        num_2_val = float(num_2.get())

        # Set the value of solution to update
        # the entry box
        solution.set(num_1_val + num_2_val)
    except ValueError:
        pass


# Create the main window that holds all the widgets
root = Tk()

# Define the title for the window
root.title("Calculator")

# The frame surrounds the interface with the widgets
# A frame is used so the widgets and background
# colors are consistent
# Define padding for left top and right bottom
frame = ttk.Frame(root, padding="10 10 10 10")

# ---------- GRID GEOMETRY MANAGER  ----------
# The Grid manager is the most useful using a series
# of rows and columns for laying out widgets

# Each cell can only hold 1 widget, but a widget
# can cover multiple cells.

# rows start at 0, 1, ...
# columns start at 0, 1, ...

# sticky defines how the widget expands (N, NE, E, SE,
# S, SW, W, NW)

# Define that a grid should stick to the North, West,
# East and South sides of the frame
frame.grid(column=0, row=0, sticky=(N, W, E, S))

# Define that the frame should expand with the main window
# If columns and rows have the same weight they will
# expand at the same rate when the interface is expanded
root.columnconfigure(0, weight=1)
root.rowconfigure(0, weight=1)

# Define Tkinter string variables
num_1 = StringVar()
```

```
num_2 = StringVar()
solution = StringVar()

# Create entry box 7 characters long that has the value
# entered assigned to num_1
num_1_entry = ttk.Entry(frame, width=7,
                        textvariable=num_1)

# Place in the 1st column, 1st row
# W E means that the widget should expand horizontally
# with the surrounding interface
num_1_entry.grid(column=1, row=1, sticky=(W, E))

# Place a label with the value + in the 2nd column
ttk.Label(frame, text="+").grid(column=2, row=1,
                                sticky=(W, E))

# Create 2nd number entry box
num_2_entry = ttk.Entry(frame, width=7,
                        textvariable=num_2)
num_2_entry.grid(column=3, row=1, sticky=(W, E))

ttk.Button(frame, text="Add", command=get_sum).grid(column=1, row=2, sticky=W)

solution_entry = ttk.Entry(frame, width=7, textvariable=solution)
solution_entry.grid(column=3, row=2, sticky=(W, E))

# Put focus on the num_1 entry box
num_1_entry.focus()

# When the return button is pressed call the function calculate
root.bind('<Return>', get_sum)

# A loop that executes until the application exits
root.mainloop()
```

 # ---------- Python Tkinter Tutorial 2 ----------

```
# Here I'll show how to use the widgets label, entry
# button, check button, radio buttons and combo boxes

# Get the standard library for Tk
from tkinter import *

# Get the newest widget themes from Tk 8.5
from tkinter import ttk


def set_entry(*args):
    entry_1_txt.set("Hello")


def chk_but_changed(*args):
```

```
    entry_1_txt.set(chk_but_1_txt.get())


def radio_changed(*args):
    entry_1_txt.set(radio_but_1_val.get())


def combo_changed(*args):
    entry_1_txt.set(combo_1_val.get())


root = Tk()
root.title("Widget Example")

frame = ttk.Frame(root, padding="10 10 10 10")

# ---------- GRID GEOMETRY MANAGER  ----------
frame.grid(column=0, row=0, sticky=(N, W, E, S))
root.columnconfigure(0, weight=1)
root.rowconfigure(0, weight=1)

# ----- LABELS -----
# Used to get and set value of the label
label_1_txt = StringVar()
# Define the parent and text in the label
label_1 = ttk.Label(frame, text='Data :')
label_1.grid(column=1, row=1, sticky=(W, E))

# To change a value you must attach to the StringVar class
label_1['textvariable'] = label_1_txt
label_1_txt.set('Data ')

# ----- ENTRY -----
# Used to get and set the value of the entry
entry_1_txt = StringVar()
# Define the number of characters long and the StringVar
# it is associated with
entry_1 = ttk.Entry(frame, width=7, textvariable=entry_1_txt)
entry_1.grid(column=2, row=1, sticky=(W, E))

# You can get values by using get on a StringVar
entry_1_txt.set(label_1_txt.get())

# ----- BUTTON -----
# Assign text in button and the function to call when clicked
button_1 = ttk.Button(frame, text='Click', command=set_entry)
button_1.grid(column=3, row=1, sticky=(W, E))

# You can disable the button
button_1['state'] = 'disabled'
# And, enable it again
button_1['state'] = 'enable'
```

```
# Check if it is disabled (1 = True, 0 = False)
entry_1_txt.set(button_1.instate(['disabled']))

# ----- CHECK BUTTON -----
chk_but_1_txt = StringVar()
# Text to display next to it, function to call, StringVar,
# value assigned when check and not checked
chk_but_1 = ttk.Checkbutton(frame, text='Feelings',
                        command=chk_but_changed,
                        variable=chk_but_1_txt,
                        onvalue='Happy', offvalue='Sad')
chk_but_1.grid(column=4, row=1, sticky=(W, E))

# ----- RADIO BUTTONS -----
# Only 1 radio button can be selected at a time
# Shared StringVar
radio_but_1_val = StringVar()
# Parent, text assigned, StringVar, value assigned to StringVar,
# function to call on event
red_r_but = ttk.Radiobutton(frame, text='Red',
                        variable=radio_but_1_val,
                        value='Red', command=radio_changed)
blue_r_but = ttk.Radiobutton(frame, text='Blue',
                        variable=radio_but_1_val,
                        value='Blue', command=radio_changed)
green_r_but = ttk.Radiobutton(frame, text='Green',
                        variable=radio_but_1_val,
                        value='Green', command=radio_changed)
red_r_but.grid(column=2, row=2, sticky=(W, E))
blue_r_but.grid(column=3, row=2, sticky=(W, E))
green_r_but.grid(column=4, row=2, sticky=(W, E))

# Label for radio buttons
label_2 = ttk.Label(frame, text='Fav Color')
label_2.grid(column=1, row=2, sticky=(W, E))

# ----- COMBOBOX -----
# Drop down boxes that contain a list of values
combo_1_val = StringVar()
combo_1 = ttk.Combobox(frame, textvariable=combo_1_val)
label_3 = ttk.Label(frame, text='Size')
label_3.grid(column=1, row=3, sticky=(W, E))

# Assign values to the combobox
combo_1['values'] = ('Small', 'Medium', 'Large')
combo_1.grid(column=2, row=3, sticky=(W, E))

# Call a function when combobox is changed
combo_1.bind('<<ComboboxSelected>>', combo_changed)

# A loop that executes until the application exits
root.mainloop()
```

```
# ---------------------------------- Python Tkinter Tutorial 3 ---------------------------------

from tkinter import *
from tkinter import ttk


class Calculator:

    def __init__(self, root):
        # Will hold the changing value stored in the entry
        self.entry_value = StringVar(root, value="")

        # Define title for the app
        root.title("Calculator")

        # Defines the width and height of the window
        root.geometry("483x220")

        # Block resizing of Window
        root.resizable(width=False, height=False)

        # Customize the styling for the buttons and entry
        style = ttk.Style()
        style.configure("TButton",
                        font="Serif 15",
                        padding=10)

        style.configure("TEntry",
                        font="Serif 18",
                        padding=10)

        # Create the text entry box
        self.number_entry = ttk.Entry(root,
                                      textvariable=self.entry_value, width=50)
        self.number_entry.grid(row=0, columnspan=4, sticky=(W, E))

        # ----- 1st Row -----

        self.button7 = ttk.Button(root, text="7").grid(row=1, column=0, sticky=(W, E))

        self.button8 = ttk.Button(root, text="8").grid(row=1, column=1, sticky=(W, E))

        self.button9 = ttk.Button(root, text="9").grid(row=1, column=2, sticky=(W, E))

        self.button_div = ttk.Button(root, text="/").grid(row=1, column=3, sticky=(W, E))

        # ----- 2nd Row -----

        self.button4 = ttk.Button(root, text="4").grid(row=2, column=0, sticky=(W, E))

        self.button5 = ttk.Button(root, text="5").grid(row=2, column=1, sticky=(W, E))

        self.button6 = ttk.Button(root, text="6").grid(row=2, column=2, sticky=(W, E))
```

```
        self.button_mult = ttk.Button(root, text="*").grid(row=2, column=3, sticky=(W, E))

        # ----- 3rd Row -----

        self.button1 = ttk.Button(root, text="1").grid(row=3, column=0, sticky=(W, E))

        self.button2 = ttk.Button(root, text="2").grid(row=3, column=1, sticky=(W, E))

        self.button3 = ttk.Button(root, text="3").grid(row=3, column=2, sticky=(W, E))

        self.button_add = ttk.Button(root, text="+").grid(row=3, column=3, sticky=(W, E))

        # ----- 4th Row -----

        self.button_clear = ttk.Button(root, text="AC").grid(row=4, column=0, sticky=(W, E))

        self.button0 = ttk.Button(root, text="0").grid(row=4, column=1, sticky=(W, E))

        self.button_equal = ttk.Button(root, text="=").grid(row=4, column=2, sticky=(W, E))

        self.button_sub = ttk.Button(root, text="-").grid(row=4, column=3, sticky=(W, E))


# Get the root window object
root = Tk()

# Create the calculator
calc = Calculator(root)

# Run the app until exited
root.mainloop()

# ---------------------------------- Python Tkinter Tutorial 4 ---------------------------------

from tkinter import *
from tkinter import ttk

'''
USE CASE

1. User clicks a number
   a. Get current value in entry widget
   b. Put new value to the right of current value
   c. Clear the entry box
   d. Insert the new number
   * Note Start entry as clear

2. User clicks a math button
   a. Check if there is a value in entry
   b. Store which math button was pressed
   c. Call the matching math calculation based on button press
   d. Store current entry value
```

```
    e. Clear entry widget
    f. Prepare for next entry so a calculation can be made
    * Note handle conversion of values to floats

3. User clicks the equal button
    a. Check if a math button has been clicked
    b. Check which math button was clicked last
    c. Get the stored 1st number value entered
    d. Get the value currently in the entry widget
    e. Perform the correct calculation
    f. Clear entry widget
    g. Place the new calculation solution in entry

4. User clicks AC
    a. Clear all math button presses
    b. Clear values in entry widget
    c. Put a 0 in entry widget
    d. Store 0 in current value

'''


class Calculator:
    # Stores the current value to display in the entry
    calc_value = 0.0

    # Will define if this was the last math button clicked
    div_trigger = False
    mult_trigger = False
    add_trigger = False
    sub_trigger = False

    # Called anytime a number button is pressed
    def button_press(self, value):

        if value == 'AC':
            # make false to cancel out previous math button click
            self.add_trigger = False
            self.sub_trigger = False
            self.mult_trigger = False
            self.div_trigger = False
            # Clear the entry box
            # (0, "end") refers to all characters in the widget
            self.number_entry.delete(0, "end")
            entry_val = 0
        else:
            # Get the current value in the entry
            entry_val = self.number_entry.get()

            # Put the new value to the right of it
            # If it was 1 and 2 is pressed it is now 12
            # Otherwise the new number goes on the left
            entry_val += value
```

```
        # Clear the entry box
        self.number_entry.delete(0, "end")

        # Insert the new value going from left to right
        self.number_entry.insert(0, entry_val)

# Returns True or False if the string is a float
def is_float(self, str_val):
    try:

        # If the string isn't a float float() will throw a
        # ValueError
        float(str_val)

        # If there is a value you want to return use return
        return True
    except ValueError:
        return False

# Handles logic when math buttons are pressed
def math_button_press(self, value):

    # Only do anything if entry currently contains a number
    if self.is_float(str(self.number_entry.get())):

        # make false to cancel out previous math button click
        self.add_trigger = False
        self.sub_trigger = False
        self.mult_trigger = False
        self.div_trigger = False

        # Get the value out of the entry box for the calculation
        self.calc_value = float(self.entry_value.get())

        # Set the math button click so when equals is clicked
        # that function knows what calculation to use
        if value == "/":
            print("/ Pressed")
            self.div_trigger = True
        elif value == "*":
            print("* Pressed")
            self.mult_trigger = True
        elif value == "+":
            print("+ Pressed")
            self.add_trigger = True
        else:
            print("- Pressed")
            self.sub_trigger = True

        # Clear the entry box
        self.number_entry.delete(0, "end")

# Performs a mathematical operation by taking the value before
# the math button is clicked and the current value. Then perform
```

```
# the right calculation by checking what math button was clicked
# last
def equal_button_press(self):

    # Make sure a math button was clicked
    if self.add_trigger or self.sub_trigger or self.mult_trigger or self.div_trigger:

        if self.add_trigger:
            solution = self.calc_value + float(self.entry_value.get())
        elif self.sub_trigger:
            solution = self.calc_value - float(self.entry_value.get())
        elif self.mult_trigger:
            solution = self.calc_value * float(self.entry_value.get())
        else:
            solution = self.calc_value / float(self.entry_value.get())

        print(self.calc_value, " ", float(self.entry_value.get()),
              " ", solution)

        # Clear the entry box
        self.number_entry.delete(0, "end")

        self.number_entry.insert(0, solution)

def __init__(self, root):
    # Will hold the changing value stored in the entry
    self.entry_value = StringVar(root, value="")

    # Define title for the app
    root.title("Calculator")

    # Defines the width and height of the window
    root.geometry("483x220")

    # Block resizing of Window
    root.resizable(width=False, height=False)

    # Customize the styling for the buttons and entry
    style = ttk.Style()
    style.configure("TButton",
                    font="Serif 15",
                    padding=10)

    style.configure("TEntry",
                    font="Serif 18",
                    padding=10)

    # Create the text entry box
    self.number_entry = ttk.Entry(root,
                                  textvariable=self.entry_value, width=50)
    self.number_entry.grid(row=0, columnspan=4, sticky=(W, E))

    # ----- 1st Row -----
```

```
        self.button7 = ttk.Button(root, text="7", command=lambda:
self.button_press('7')).grid(row=1, column=0,
                                                                sticky=(W, E))

        self.button8 = ttk.Button(root, text="8", command=lambda:
self.button_press('8')).grid(row=1, column=1,
                                                                sticky=(W, E))

        self.button9 = ttk.Button(root, text="9", command=lambda:
self.button_press('9')).grid(row=1, column=2,
                                                                sticky=(W, E))

        self.button_div = ttk.Button(root, text="/", command=lambda:
self.math_button_press('/')).grid(row=1, column=3,
                                                                    sticky=(W, E))

        # ----- 2nd Row -----

        self.button4 = ttk.Button(root, text="4", command=lambda:
self.button_press('4')).grid(row=2, column=0,
                                                                sticky=(W, E))

        self.button5 = ttk.Button(root, text="5", command=lambda:
self.button_press('5')).grid(row=2, column=1,
                                                                sticky=(W, E))

        self.button6 = ttk.Button(root, text="6", command=lambda:
self.button_press('6')).grid(row=2, column=2,
                                                                sticky=(W, E))

        self.button_mult = ttk.Button(root, text="*", command=lambda:
self.math_button_press('*')).grid(row=2, column=3,
                                                                    sticky=(W, E))

        # ----- 3rd Row -----

        self.button1 = ttk.Button(root, text="1", command=lambda:
self.button_press('1')).grid(row=3, column=0,
                                                                sticky=(W, E))

        self.button2 = ttk.Button(root, text="2", command=lambda:
self.button_press('2')).grid(row=3, column=1,
                                                                sticky=(W, E))

        self.button3 = ttk.Button(root, text="3", command=lambda:
self.button_press('3')).grid(row=3, column=2,
                                                                sticky=(W, E))

        self.button_add = ttk.Button(root, text="+", command=lambda:
self.math_button_press('+')).grid(row=3, column=3,
                                                                    sticky=(W, E))

        # ----- 4th Row -----
```

```
        self.button_clear = ttk.Button(root, text="AC", command=lambda:
self.button_press('AC')).grid(row=4, column=0,
                                                                sticky=(W, E))

        self.button0 = ttk.Button(root, text="0", command=lambda:
self.button_press('0')).grid(row=4, column=1,
                                                            sticky=(W, E))

        self.button_equal = ttk.Button(root, text="=", command=lambda:
self.equal_button_press()).grid(row=4, column=2,
                                                                sticky=(W, E))

        self.button_sub = ttk.Button(root, text="-", command=lambda:
self.math_button_press('-')).grid(row=4, column=3,
                                                                sticky=(W, E))


# Get the root window object
root = Tk()

# Create the calculator
calc = Calculator(root)

# Run the app until exited
root.mainloop()
```

 # ---------- TKINTER MENU BAR TUTORIAL ----------

```
from tkinter import *
from tkinter import messagebox
from tkinter import ttk

# In this tutorial I'll cover
# Creating menu bars
# Triggering functions from the menu bar
# Using Checkboxes & Radio Buttons
# Adding shortcut keys to menu bar items

# When called this closes the app
def quit_app():
    root.quit()


# Shows an about message box
def show_about(event=None):
    messagebox.showwarning(
        "About",
        "Isn't this an awesome program?"
    )


# Output to console when the font is changed
def change_font(event=None):
```

```
    print("Font Changed to", text_font.get())


# Creates the main window
root = Tk()

# Add a title for your app
root.title("Menu Bar Example")

# Create the menu bar
the_menu = Menu(root)

# ----- FILE MENU ITEMS -----
# Create a pull down menu that can't be removed
# and tie it to your menu bar
file_menu = Menu(the_menu, tearoff=0)

# Add items that will show up when File is clicked on
file_menu.add_command(label="Open")
file_menu.add_command(label="Save")

# Add horizontal bar between items in menu
file_menu.add_separator()

# Add Quit to the menu and execute the function
# that quits the app
file_menu.add_command(label="Quit", command=quit_app)

# Add the label File and the pull down menu to
# the menu bar
the_menu.add_cascade(label="File", menu=file_menu)
# ----- END OF FILE MENU ITEMS -----

# ----- VIEW MENU ITEMS -----
# Create pull down for View
view_menu = Menu(the_menu, tearoff=0)

# Add a checkbox option to show line numbers and
# set the default to checked
line_numbers = IntVar()
line_numbers.set(1)

# Add checkbutton to View with a label and
# bind line_numbers so we know if the box is
# checked or not
view_menu.add_checkbutton(label="Line Numbers",
                          variable=line_numbers)

# Add View to the menu
the_menu.add_cascade(label="View", menu=view_menu)
# ----- END OF VIEW MENU ITEMS -----

# ----- FONT MENU ITEMS -----
# Store the font chosen in a string variable
```

```
text_font = StringVar()

# Set the default font
text_font.set("Times")

# Create pull down for font
font_menu = Menu(the_menu, tearoff=0)

# Define radio buttons for the menu, store selection
# in text_font and call change_font when changed
font_menu.add_radiobutton(label="Times",
                          variable=text_font,
                          command=change_font)
font_menu.add_radiobutton(label="Courier",
                          variable=text_font,
                          command=change_font)
font_menu.add_radiobutton(label="Arial",
                          variable=text_font,
                          command=change_font)

# Add Font to the menu
the_menu.add_cascade(label="Font", menu=font_menu)
# ----- END OF FONT MENU ITEMS -----

# ----- HELP MENU ITEMS -----
# Create pull down for Help
help_menu = Menu(the_menu, tearoff=0)

# When About is clicked execute a function but
# also tie it to a shortcut
# Accelerator defines a shortcut that's available
help_menu.add_command(label="About",
                      accelerator="command-A",
                      command=show_about)

# Key substitutions on Windows and Mac
# Control (Windows) = Command (Mac)
# Alt (Windows) = Option (Mac)

# Add Help to the menu bar
the_menu.add_cascade(label="Help", menu=help_menu)

# Bind shortcut to the About and what we want
# to show
root.bind('<Command-A>', show_about)

# Also bind lowercase a
root.bind('<Command-a>', show_about)

# ----- END OF HELP MENU ITEMS -----

# Make your menu show on the screen
root.config(menu=the_menu)
```

```
# Keeps our program running until quit
root.mainloop()

# ---------- TKINTER TEXT EDITOR 1 ----------

from tkinter import *
import tkinter.filedialog


class TextEditor:

    # Quits the TkInter app when called
    @staticmethod
    def quit_app(event=None):
        root.quit()

    # ----- NEXT TUTORIAL -----

    def remake_file(self, text_area_list):
        for i in text_area_list:
            print("Key", i[0])
            print("Value", i[1])
            print("Index", i[2])


    # ----- END NEXT TUTORIAL -----

    def open_file(self, event=None):
        # Open dialog and get chosen file
        txt_file = tkinter.filedialog.askopenfilename(parent=root,
                                              initialdir='/')
        # If the file exists
        if txt_file:

            self.text_area.delete(1.0, END)

            # Open file and put text in the text widget
            with open(txt_file) as _file:
                self.text_area.insert(1.0, _file.read())

                # Update the text widget
                root.update_idletasks()

    def save_file(self, event=None):

        # Opens the save as dialog box
        file = tkinter.filedialog.asksaveasfile(mode='w')
        if file is not None:
            # Get text in the text widget and delete the last newline
            data = self.text_area.get('1.0', END + '-1c')

            # Write the text and close
```

```
        file.write(data)

        # ----- NEXT TUTORIAL -----

        print(str(self.text_area.dump('1.0', END)))
        self.remake_file(self.text_area.dump('1.0', END))

        # ----- END NEXT TUTORIAL -----

        file.close()

    def make_bold(self):
        self.text_area.tag_add("bt", "sel.first", "sel.last")

    def __init__(self, root):

        self.text_to_write = ""

        # Define title for the app
        root.title("Text Editor")

        # Defines the width and height of the window
        root.geometry("600x550")

        frame = Frame(root, width=600, height=550)

        # Create the scrollbar
        scrollbar = Scrollbar(frame)

        # yscrollcommand connects the scroll bar to the text
        # area
        self.text_area = Text(frame, width=600, height=550,
                              yscrollcommand=scrollbar.set,
                              padx=10, pady=10, font=("Georgia", "28"))

        # Call yview when the scrollbar is moved
        scrollbar.config(command=self.text_area.yview)

        # Put scroll bar on the right and fill in the Y direction
        scrollbar.pack(side="right", fill="y")

        # Pack on the left and fill available space
        self.text_area.pack(side="left", fill="both", expand=True)
        frame.pack()

        # ----- FILE MENU CREATION -----

        # Create a pull down menu that can't be removed
        file_menu = Menu(the_menu, tearoff=0)

        # Add items to the menu that show when clicked
        # compound allows you to add an image
        file_menu.add_command(label="Open", command=self.open_file)
        file_menu.add_command(label="Save", command=self.save_file)
```

```
        # Add a horizontal bar to group similar commands
        file_menu.add_separator()

        # Call for the function to execute when clicked
        file_menu.add_command(label="Quit", command=self.quit_app)

        # Add the pull down menu to the menu bar
        the_menu.add_cascade(label="File", menu=file_menu)

        # ----- EDIT MENU CREATION -----

        edit_menu = Menu(the_menu, tearoff=0)
        edit_menu.add_command(label="Bold", command=self.make_bold)
        the_menu.add_cascade(label="Edit", menu=edit_menu)

        self.text_area.tag_config("bt", font=("Georgia", "28", "bold"))


        # Display the menu bar
        root.config(menu=the_menu)


root = Tk()

# Create the menu object
the_menu = Menu(root)

text_editor = TextEditor(root)

root.mainloop()
```

 # ---------- TKINTER TEXT EDITOR 2 ----------

```
from tkinter import *
import tkinter.filedialog
import ast


class TextEditor:

    # Quits the TkInter app when called
    @staticmethod
    def quit_app(event=None):
        root.quit()

    # ----- NEXT TUTORIAL -----

    def remake_file(self, text_area_list):
        for i in text_area_list:
            print("Key", i[0])
            print("Value", i[1])
            print("Index", i[2])
```

```
# ----- END NEXT TUTORIAL -----

def open_file(self, event=None):
    # Open dialog and get chosen file
    txt_file = tkinter.filedialog.askopenfilename(parent=root,
                                        initialdir='/')
    # If the file exists
    if txt_file:
        self.text_area.delete(1.0, END)

        # Holds list of tuples
        file_list = []

        # Open file and put text in the text widget
        with open(txt_file) as _file:
            # self.text_area.insert(1.0, _file.read())

            # Processes the list of tuples into a list
            file_list = list(ast.literal_eval(_file.read()))
            print(file_list)

            # Search for text in the list and put it in the right
            # index position
            for data in file_list:
                if data[0] == "text":
                    self.text_area.insert(data[2], data[1])

            # Cycle through the list looking for tagon, but ignore sel
            i = 0
            while i < len(file_list):
                if (file_list[i][0] == "tagon") and (file_list[i][1] != "sel"):
                    # Get the styling tag
                    styling = file_list[i][1]
                    # Get the index where styling begins
                    start_of_style = file_list[i][2]
                    # Used to get the index where styling ends
                    # but set as end of file by default
                    end_of_style = END
                    # Make sure I'm not searching beyond the end
                    # of the list
                    if (i+3) < len(file_list):
                        # If not find the end index
                        end_of_style = file_list[i+4][2]
                    print("Style", styling)
                    print("Start", start_of_style)
                    print("End", end_of_style)
                    # Add styling provided along with the start
                    # and ending index
                    self.text_area.tag_add(styling,
                                           start_of_style,
                                           end_of_style)
                i += 1
```

```
            # Update the text widget
            root.update_idletasks()

    def save_file(self, event=None):

        # Opens the save as dialog box
        file = tkinter.filedialog.asksaveasfile(mode='w')
        if file is not None:
            # Get text in the text widget and delete the last newline
            data = self.text_area.get('1.0', END + '-1c')

            # Write the text and close
            # file.write(data)

            # ----- NEXT TUTORIAL -----

            # print(str(self.text_area.dump('1.0', END)))
            # self.remake_file(self.text_area.dump('1.0', END))

            # Get list of tuples
            text_area_list = self.text_area.dump('1.0', END + '-1c')
            # Write list of tuples to file
            file.write(' '.join('("{}", "{}", "{}")'.format(x[0],
                                                            x[1], x[2])
                                for x in text_area_list))

            # ----- END NEXT TUTORIAL -----

            file.close()

    def make_bold(self):
        self.text_area.tag_add("bt", "sel.first", "sel.last")

    def __init__(self, root):

        self.text_to_write = ""

        # Define title for the app
        root.title("Text Editor")

        # Defines the width and height of the window
        root.geometry("600x550")

        frame = Frame(root, width=600, height=550)

        # Create the scrollbar
        scrollbar = Scrollbar(frame)

        # yscrollcommand connects the scroll bar to the text
        # area
        self.text_area = Text(frame, width=600, height=550,
                              yscrollcommand=scrollbar.set,
                              padx=10, pady=10, font=("Georgia", "14"))
```

```
        # Call yview when the scrollbar is moved
        scrollbar.config(command=self.text_area.yview)

        # Put scroll bar on the right and fill in the Y direction
        scrollbar.pack(side="right", fill="y")

        # Pack on the left and fill available space
        self.text_area.pack(side="left", fill="both", expand=True)
        frame.pack()

        # ----- FILE MENU CREATION -----

        # Create a pull down menu that can't be removed
        file_menu = Menu(the_menu, tearoff=0)

        # Add items to the menu that show when clicked
        # compound allows you to add an image
        file_menu.add_command(label="Open", command=self.open_file)
        file_menu.add_command(label="Save", command=self.save_file)

        # Add a horizontal bar to group similar commands
        file_menu.add_separator()

        # Call for the function to execute when clicked
        file_menu.add_command(label="Quit", command=self.quit_app)

        # Add the pull down menu to the menu bar
        the_menu.add_cascade(label="File", menu=file_menu)

        # ----- EDIT MENU CREATION -----

        edit_menu = Menu(the_menu, tearoff=0)
        edit_menu.add_command(label="Bold", command=self.make_bold)
        the_menu.add_cascade(label="Edit", menu=edit_menu)

        self.text_area.tag_config("bt", font=("Georgia", "14", "bold"))

        # Display the menu bar
        root.config(menu=the_menu)


root = Tk()

# Create the menu object
the_menu = Menu(root)

text_editor = TextEditor(root)

root.mainloop()
```

```
# ---------- TKINTER TEXT EDITOR 3 ----------

from tkinter import *
import tkinter.filedialog
import ast

# NEW Type from Pillow import Image, ImageTkClick
# and let PyCharm install Pillow
# Image allows you to load images from files
# ImageTk provides ways to create and modify images
from PIL import Image, ImageTk
# --- END NEW ---


class TextEditor:
    # NEW
    # Used for font size, type
    font_size = 28
    font_type = "Georgia"
    # END NEW

    # Quits the TkInter app when called
    @staticmethod
    def quit_app(event=None):
        root.quit()

    def open_file(self, event=None):
        # Open dialog and get chosen file
        txt_file = tkinter.filedialog.askopenfilename(parent=root)
        # If the file exists
        if txt_file:
            self.text_area.delete(1.0, END)

            # Holds list of tuples
            file_list = []

            # Open file and put text in the text widget
            with open(txt_file) as _file:
                # self.text_area.insert(1.0, _file.read())

                # Processes the list of tuples into a list
                file_list = list(ast.literal_eval(_file.read()))
                print(file_list)

                # Search for text in the list and put it in the right
                # index position
                for data in file_list:
                    if data[0] == "text":
                        self.text_area.insert(data[2], data[1])

                # Cycle through the list looking for tagon, but ignore sel
                i = 0
                while i < len(file_list):
```

```
            # NEW Remove the sel option
            if file_list[i][0] == "tagon":
                # Get the styling tag
                styling = file_list[i][1]
                # Get the index where styling begins
                start_of_style = file_list[i][2]
                # Used to get the index where styling ends
                # but set as end of file by default
                end_of_style = END
                # Make sure I'm not searching beyond the end
                # of the list

                # NEW Change the step to 2 because
                # we got rid of sel and mark
                if (i+2) < len(file_list):
                    # If not find the end index
                    end_of_style = file_list[i+2][2]

                # Add styling provided along with the start
                # and ending index
                self.text_area.tag_add(styling,
                                       start_of_style,
                                       end_of_style)
            i += 1

        # Update the text widget
        root.update_idletasks()

def save_file(self, event=None):

    # Opens the save as dialog box
    file = tkinter.filedialog.asksaveasfile(mode='w')
    if file is not None:
        # Get list of tuples
        text_area_list = self.text_area.dump('1.0', END + '-1c')

        # --- NEW ---
        # Remove all tuples if 'sel' or 'mark' is in it
        text_area_list = [i for i in text_area_list if i[1] != 'sel' and i[0] != 'mark']
        # --- END NEW ---

        # Write list of tuples to file
        file.write(' '.join('("{}", "{}", "{}")'.format(x[0],
                    x[1], x[2])
                    for x in text_area_list))
        file.close()

def make_bold(self):
    self.text_area.tag_add("bt", "sel.first", "sel.last")

# NEW Make selected text italic
def make_italic(self):
    self.text_area.tag_add("ital", "sel.first", "sel.last")
```

```
# --- END NEW ---

def __init__(self, root):

    self.text_to_write = ""

    # Define title for the app
    root.title("Text Editor")

    # Defines the width and height of the window
    root.geometry("600x550")

    frame = Frame(root, width=600, height=550)

    # Create the scrollbar
    scrollbar = Scrollbar(frame)

    # yscrollcommand connects the scroll bar to the text
    # area
    self.text_area = Text(frame, width=600, height=550,
                          yscrollcommand=scrollbar.set,
                          padx=10, pady=10, font=(self.font_type, self.font_size))

    # Call yview when the scrollbar is moved
    scrollbar.config(command=self.text_area.yview)

    # Put scroll bar on the right and fill in the Y direction
    scrollbar.pack(side="right", fill="y")

    # Pack on the left and fill available space
    self.text_area.pack(side="left", fill="both", expand=True)

    # NEW Moved this below the toolbar
    # frame.pack()

    # ----- FILE MENU CREATION -----

    # Create a pull down menu that can't be removed
    file_menu = Menu(the_menu, tearoff=0)

    # Add items to the menu that show when clicked
    # compound allows you to add an image
    file_menu.add_command(label="Open", command=self.open_file)
    file_menu.add_command(label="Save", command=self.save_file)

    # Add a horizontal bar to group similar commands
    file_menu.add_separator()

    # Call for the function to execute when clicked
    file_menu.add_command(label="Quit", command=self.quit_app)

    # Add the pull down menu to the menu bar
    the_menu.add_cascade(label="File", menu=file_menu)
```

```
# ----- EDIT MENU CREATION -----

edit_menu = Menu(the_menu, tearoff=0)
edit_menu.add_command(label="Bold", command=self.make_bold)

# --- NEW ---
# Add italic option to menu bar
edit_menu.add_command(label="Italic", command=self.make_italic)
# --- END NEW ---

the_menu.add_cascade(label="Edit", menu=edit_menu)

self.text_area.tag_config("bt", font=(self.font_type, self.font_size, "bold"))

# --- New Configure italic ---
self.text_area.tag_config("ital", font=(self.font_type, self.font_size, "italic"))

# Create our tool bar by creating a frame, defining the border
# width, and relief=RAISED draws a line under the toolbar
toolbar = Frame(root, bd=1, relief=RAISED)

# Get our tool bar images
open_img = Image.open("open.png")
save_img = Image.open("save.png")
copy_img = Image.open("copy.png")
cut_img = Image.open("cut.png")
paste_img = Image.open("paste.png")
bold_img = Image.open("bold.png")
italic_img = Image.open("italic.png")

# Create TkInter image to be used in buttons
open_icon = ImageTk.PhotoImage(open_img)
save_icon = ImageTk.PhotoImage(save_img)
copy_icon = ImageTk.PhotoImage(copy_img)
cut_icon = ImageTk.PhotoImage(cut_img)
paste_icon = ImageTk.PhotoImage(paste_img)
bold_icon = ImageTk.PhotoImage(bold_img)
italic_icon = ImageTk.PhotoImage(italic_img)

# Create buttons for the toolbar
open_button = Button(toolbar, image=open_icon,
                     command=self.open_file)
open_button.image = open_icon
save_button = Button(toolbar, image=save_icon,
                     command=self.save_file)
save_button.image = save_icon
copy_button = Button(toolbar, image=copy_icon,
                     command=lambda: root.focus_get().event_generate('<<Copy>>'))
copy_button.image = copy_icon
cut_button = Button(toolbar, image=cut_icon,
                    command=lambda: root.focus_get().event_generate('<<Cut>>'))
cut_button.image = cut_icon
paste_button = Button(toolbar, image=paste_icon,
                      command=lambda: root.focus_get().event_generate('<<Paste>>'))
```

```
        paste_button.image = paste_icon
        bold_button = Button(toolbar, image=bold_icon,
                             command=self.make_bold)
        bold_button.image = bold_icon
        italic_button = Button(toolbar, image=italic_icon,
                               command=self.make_italic)
        italic_button.image = italic_icon

        # Place buttons in the interface
        open_button.pack(side=LEFT, padx=2, pady=2)
        save_button.pack(side=LEFT, padx=2, pady=2)
        copy_button.pack(side=LEFT, padx=2, pady=2)
        cut_button.pack(side=LEFT, padx=2, pady=2)
        paste_button.pack(side=LEFT, padx=2, pady=2)
        bold_button.pack(side=LEFT, padx=2, pady=2)
        italic_button.pack(side=LEFT, padx=2, pady=2)

        # Put toolbar at the top of the window
        # and fill horizontally
        toolbar.pack(side=TOP, fill=X)

        # Moved from the top
        frame.pack()

        # --- END NEW ---

        # Display the menu bar
        root.config(menu=the_menu)


root = Tk()

# Create the menu object
the_menu = Menu(root)

text_editor = TextEditor(root)

root.mainloop()
```

 # ---------- TKINTER PAINT APP 1 ----------

```
from tkinter import *
import tkinter.font

# Create main window
root = Tk()
root.geometry("800x600")

class PaintApp:
    text_font = StringVar()
    text_size = IntVar()
    bold_text = IntVar()
    italic_text = IntVar()
```

```
# Stores current tool we are using
drawing_tool = StringVar()

# Tracks whether left mouse is down
left_but = "up"

# x and y positions for drawing with pencil
x_pos, y_pos = None, None

# Tracks x & y when the mouse is clicked and released
x1_line_pt, y1_line_pt, x2_line_pt, y2_line_pt = None, None, None, None

# Quits the TkInter app when called
@staticmethod
def quit_app():
    root.quit()

def make_menu_bar(self):
    # Create the menu object
    the_menu = Menu(root)

    # ---- FILE MENU ----
    # Create a pull down menu that can't be removed
    file_menu = Menu(the_menu, tearoff=0)

    # Add items to the menu that show when clicked
    # compound allows you to add an image
    file_menu.add_command(label="Open")
    file_menu.add_command(label="Save")

    # Add a horizontal bar to group similar commands
    file_menu.add_separator()

    # Call for the function to execute when clicked
    file_menu.add_command(label="Quit", command=self.quit_app)

    # Add the pull down menu to the menu bar
    the_menu.add_cascade(label="File", menu=file_menu)

    # ---- FONT MENU ----
    font_menu = Menu(the_menu, tearoff=0)
    font_type_submenu = Menu(font_menu)
    font_type_submenu.add_radiobutton(label="Times",
                          variable=self.text_font)
    font_type_submenu.add_radiobutton(label="Courier",
                          variable=self.text_font)
    font_type_submenu.add_radiobutton(label="Ariel",
                          variable=self.text_font)
    font_menu.add_cascade(label="Font Type",
                      menu=font_type_submenu)

    font_size_submenu = Menu(font_menu)
    font_size_submenu.add_radiobutton(label="10",
                                variable=self.text_size,
```

```
                                value=10)
        font_size_submenu.add_radiobutton(label="15",
                                variable=self.text_size,
                                value=15)
        font_size_submenu.add_radiobutton(label="20",
                                variable=self.text_size,
                                value=20)
        font_size_submenu.add_radiobutton(label="25",
                                variable=self.text_size,
                                value=25)
        font_menu.add_cascade(label="Font Size",
                              menu=font_size_submenu)
        font_menu.add_checkbutton(label="Bold",
                                  variable=self.bold_text,
                                  onvalue=1,
                                  offvalue=0)
        font_menu.add_checkbutton(label="Italic",
                                  variable=self.italic_text,
                                  onvalue=1,
                                  offvalue=0)
        the_menu.add_cascade(label="Font", menu=font_menu)

        # ---- TOOL MENU ----
        tool_menu = Menu(the_menu, tearoff=0)
        tool_menu.add_radiobutton(label="Pencil",
                              variable=self.drawing_tool,
                              value="pencil")
        tool_menu.add_radiobutton(label="Line",
                              variable=self.drawing_tool,
                              value="line")
        tool_menu.add_radiobutton(label="Arc",
                              variable=self.drawing_tool,
                              value="arc")
        tool_menu.add_radiobutton(label="Oval",
                              variable=self.drawing_tool,
                              value="oval")
        tool_menu.add_radiobutton(label="Rectangle",
                              variable=self.drawing_tool,
                              value="rectangle")
        tool_menu.add_radiobutton(label="Text",
                              variable=self.drawing_tool,
                              value="text")
        the_menu.add_cascade(label="Tool", menu=tool_menu)

        # Display the menu bar
        root.config(menu=the_menu)

    def __init__(self, root):
        drawing_area = Canvas(root)
        drawing_area.pack()

        self.text_font.set("Times")
        self.text_size.set(20)
        self.bold_text.set(0)
```

```
        self.italic_text.set(0)
        self.drawing_tool.set("line")

        self.make_menu_bar()


paint_app = PaintApp(root)
root.mainloop()
```

 # ---------- TKINTER PAINT APP 2 ----------

```
from tkinter import *
import tkinter.font
from tkinter.colorchooser import *

# Create main window
root = Tk()
root.geometry("800x600")


class PaintApp:
    text_font = StringVar()
    text_size = IntVar()
    bold_text = IntVar()
    italic_text = IntVar()
    # Stores current tool we are using
    drawing_tool = StringVar()

    # NEW STORE DRAWING SETTINGS
    stroke_size = IntVar()
    fill_color = StringVar()
    stroke_color = StringVar()

    # Tracks whether left mouse is down
    left_but = "up"

    # x and y positions for drawing with pencil
    x_pos, y_pos = None, None

    # Tracks x & y when the mouse is clicked and released
    x1_line_pt, y1_line_pt, x2_line_pt, y2_line_pt = None, None, None, None

    # Quits the TkInter app when called
    @staticmethod
    def quit_app():
        root.quit()

    def make_menu_bar(self):
        # Create the menu object
        the_menu = Menu(root)

        # ---- FILE MENU ----
        # Create a pull down menu that can't be removed
```

```
file_menu = Menu(the_menu, tearoff=0)

# Add items to the menu that show when clicked
# compound allows you to add an image
file_menu.add_command(label="Open")
file_menu.add_command(label="Save")

# Add a horizontal bar to group similar commands
file_menu.add_separator()

# Call for the function to execute when clicked
file_menu.add_command(label="Quit", command=self.quit_app)

# Add the pull down menu to the menu bar
the_menu.add_cascade(label="File", menu=file_menu)

# ---- FONT MENU ----
font_menu = Menu(the_menu, tearoff=0)
font_type_submenu = Menu(font_menu)
font_type_submenu.add_radiobutton(label="Times",
                    variable=self.text_font)
font_type_submenu.add_radiobutton(label="Courier",
                    variable=self.text_font)
font_type_submenu.add_radiobutton(label="Ariel",
                    variable=self.text_font)
font_menu.add_cascade(label="Font Type",
                  menu=font_type_submenu)

font_size_submenu = Menu(font_menu)
font_size_submenu.add_radiobutton(label="10",
                            variable=self.text_size,
                            value=10)
font_size_submenu.add_radiobutton(label="15",
                            variable=self.text_size,
                            value=15)
font_size_submenu.add_radiobutton(label="20",
                            variable=self.text_size,
                            value=20)
font_size_submenu.add_radiobutton(label="25",
                            variable=self.text_size,
                            value=25)
font_menu.add_cascade(label="Font Size",
                  menu=font_size_submenu)
font_menu.add_checkbutton(label="Bold",
                    variable=self.bold_text,
                    onvalue=1,
                    offvalue=0)
font_menu.add_checkbutton(label="Italic",
                    variable=self.italic_text,
                    onvalue=1,
                    offvalue=0)
the_menu.add_cascade(label="Font", menu=font_menu)

# ---- TOOL MENU ----
```

```
    tool_menu = Menu(the_menu, tearoff=0)
    tool_menu.add_radiobutton(label="Pencil",
                              variable=self.drawing_tool,
                              value="pencil")
    tool_menu.add_radiobutton(label="Line",
                              variable=self.drawing_tool,
                              value="line")
    tool_menu.add_radiobutton(label="Arc",
                              variable=self.drawing_tool,
                              value="arc")
    tool_menu.add_radiobutton(label="Oval",
                              variable=self.drawing_tool,
                              value="oval")
    tool_menu.add_radiobutton(label="Rectangle",
                              variable=self.drawing_tool,
                              value="rectangle")
    tool_menu.add_radiobutton(label="Text",
                              variable=self.drawing_tool,
                              value="text")
    the_menu.add_cascade(label="Tool", menu=tool_menu)

    # ---- NEW COLOR MENU ----

    color_menu = Menu(the_menu, tearoff=0)
    color_menu.add_command(label="Fill", command=self.pick_fill)
    color_menu.add_command(label="Stroke", command=self.pick_stroke)
    stroke_width_submenu = Menu(color_menu)
    stroke_width_submenu.add_radiobutton(label="2",
                                variable=self.stroke_size,
                                value=2)
    stroke_width_submenu.add_radiobutton(label="3",
                                  variable=self.stroke_size,
                                  value=3)
    stroke_width_submenu.add_radiobutton(label="4",
                                  variable=self.stroke_size,
                                  value=4)
    stroke_width_submenu.add_radiobutton(label="5",
                                  variable=self.stroke_size,
                                  value=5)
    color_menu.add_cascade(label="Stroke Size",
                           menu=stroke_width_submenu)
    the_menu.add_cascade(label="Color", menu=color_menu)

    # ---- END OF NEW COLOR MENU ----

    # Display the menu bar
    root.config(menu=the_menu)

# ----- NEW STUFF -----

# ---------- CATCH MOUSE UP ----------

def left_but_down(self, event=None):
    self.left_but = "down"
```

```
        # Set x & y when mouse is clicked
        self.x1_line_pt = event.x
        self.y1_line_pt = event.y

    # ---------- CATCH MOUSE UP ----------

    def left_but_up(self, event=None):
        self.left_but = "up"

        # Reset the line
        self.x_pos = None
        self.y_pos = None

        # Set x & y when mouse is released
        self.x2_line_pt = event.x
        self.y2_line_pt = event.y

        # If mouse is released and line tool is selected
        # draw the line
        if self.drawing_tool.get() == "line":
            self.line_draw(event)
        elif self.drawing_tool.get() == "arc":
            self.arc_draw(event)
        elif self.drawing_tool.get() == "oval":
            self.oval_draw(event)
        elif self.drawing_tool.get() == "rectangle":
            self.rectangle_draw(event)
        elif self.drawing_tool.get() == "text":
            self.text_draw(event)

    # ---------- CATCH MOUSE MOVEMENT ----------

    def motion(self, event=None):
        if self.drawing_tool.get() == "pencil":
            self.pencil_draw(event)

    # ---------- DRAW PENCIL ----------

    def pencil_draw(self, event=None):
        if self.left_but == "down":

            # Make sure x and y have a value
            if self.x_pos is not None and self.y_pos is not None:
                event.widget.create_line(self.x_pos, self.y_pos, event.x, event.y, smooth=TRUE,
fill=self.stroke_color.get(), width=self.stroke_size.get())

            self.x_pos = event.x
            self.y_pos = event.y

    def line_draw(self, event=None):
        pass

    def arc_draw(self, event=None):
```

```
        pass

    def oval_draw(self, event=None):
        pass

    def rectangle_draw(self, event=None):
        pass

    def text_draw(self, event=None):
        pass

    def pick_fill(self, event=None):
        fill_color = askcolor(title='Pick Fill color')
        if None not in fill_color:
            self.fill_color.set(fill_color[1])
            print("Color ", self.fill_color.get())

    def pick_stroke(self, event=None):
        stroke_color = askcolor(title='Pick Stroke color')
        if None not in stroke_color:
            self.stroke_color.set(stroke_color[1])

    # ----- END OF NEW STUFF -----

    def __init__(self, root):
        drawing_area = Canvas(root, width=800, height=600)
        drawing_area.pack()

        self.text_font.set("Times")
        self.text_size.set(20)
        self.bold_text.set(0)
        self.italic_text.set(0)
        self.drawing_tool.set("pencil")

        # NEW COLOR DRAWING SETTINGS
        self.stroke_size.set(3)
        self.fill_color.set('#000000')
        self.stroke_color.set('#000000')

        self.make_menu_bar()

        # Set focus for catching events to the canvas
        drawing_area.focus_force()

        # NEW Assign different events to method calls
        drawing_area.bind("<Motion>", self.motion)
        drawing_area.bind("<ButtonPress-1>", self.left_but_down)
        drawing_area.bind("<ButtonRelease-1>", self.left_but_up)


paint_app = PaintApp(root)
root.mainloop()
```

```
# ---------- TKINTER PAINT APP 3 ----------

from tkinter import *
import tkinter.font
from tkinter.colorchooser import *
from tkinter import simpledialog

# Create main window
root = Tk()
root.geometry("800x600")


class PaintApp:
    text_font = StringVar()
    text_size = IntVar()

    # FIX BOLD AND ITALIC VARIABLES
    bold_text = StringVar()
    italic_text = StringVar()

    # Stores current tool we are using
    drawing_tool = StringVar()

    # STORE DRAWING SETTINGS
    stroke_size = IntVar()
    fill_color = StringVar()
    stroke_color = StringVar()

    # Tracks whether left mouse is down
    left_but = "up"

    # x and y positions for drawing with pencil
    x_pos, y_pos = None, None

    # Tracks x & y when the mouse is clicked and released
    x1_line_pt, y1_line_pt, x2_line_pt, y2_line_pt = None, None, None, None

    # Quits the TkInter app when called
    @staticmethod
    def quit_app():
        root.quit()

    def make_menu_bar(self):
        # Create the menu object
        the_menu = Menu(root)

        # ---- FILE MENU ----
        # Create a pull down menu that can't be removed
        file_menu = Menu(the_menu, tearoff=0)

        # Add items to the menu that show when clicked
        # compound allows you to add an image
        file_menu.add_command(label="Open")
```

```
file_menu.add_command(label="Save")

# Add a horizontal bar to group similar commands
file_menu.add_separator()

# Call for the function to execute when clicked
file_menu.add_command(label="Quit", command=self.quit_app)

# Add the pull down menu to the menu bar
the_menu.add_cascade(label="File", menu=file_menu)

# ---- FONT MENU ----
font_menu = Menu(the_menu, tearoff=0)
font_type_submenu = Menu(font_menu)
font_type_submenu.add_radiobutton(label="Times",
                    variable=self.text_font)
font_type_submenu.add_radiobutton(label="Courier",
                    variable=self.text_font)
font_type_submenu.add_radiobutton(label="Ariel",
                    variable=self.text_font)
font_menu.add_cascade(label="Font Type",
                  menu=font_type_submenu)

font_size_submenu = Menu(font_menu)
font_size_submenu.add_radiobutton(label="10",
                           variable=self.text_size,
                           value=10)
font_size_submenu.add_radiobutton(label="15",
                           variable=self.text_size,
                           value=15)
font_size_submenu.add_radiobutton(label="20",
                           variable=self.text_size,
                           value=20)
font_size_submenu.add_radiobutton(label="25",
                           variable=self.text_size,
                           value=25)
font_menu.add_cascade(label="Font Size",
                  menu=font_size_submenu)

# NEW FIX THE ON AND OFF VALUES FOR BOLD & ITALIC
font_menu.add_checkbutton(label="Bold",
                    variable=self.bold_text,
                    onvalue='bold',
                    offvalue='normal')
font_menu.add_checkbutton(label="Italic",
                    variable=self.italic_text,
                    onvalue='italic',
                    offvalue='roman')

the_menu.add_cascade(label="Font", menu=font_menu)

# ---- TOOL MENU ----
tool_menu = Menu(the_menu, tearoff=0)
tool_menu.add_radiobutton(label="Pencil",
```

```
                    variable=self.drawing_tool,
                    value="pencil")
    tool_menu.add_radiobutton(label="Line",
                    variable=self.drawing_tool,
                    value="line")
    tool_menu.add_radiobutton(label="Arc",
                    variable=self.drawing_tool,
                    value="arc")
    tool_menu.add_radiobutton(label="Oval",
                    variable=self.drawing_tool,
                    value="oval")
    tool_menu.add_radiobutton(label="Rectangle",
                    variable=self.drawing_tool,
                    value="rectangle")
    tool_menu.add_radiobutton(label="Text",
                    variable=self.drawing_tool,
                    value="text")
    the_menu.add_cascade(label="Tool", menu=tool_menu)

    # ---- COLOR MENU ----

    color_menu = Menu(the_menu, tearoff=0)
    color_menu.add_command(label="Fill", command=self.pick_fill)
    color_menu.add_command(label="Stroke", command=self.pick_stroke)
    stroke_width_submenu = Menu(color_menu)
    stroke_width_submenu.add_radiobutton(label="2",
                                variable=self.stroke_size,
                                value=2)
    stroke_width_submenu.add_radiobutton(label="3",
                                  variable=self.stroke_size,
                                  value=3)
    stroke_width_submenu.add_radiobutton(label="4",
                                  variable=self.stroke_size,
                                  value=4)
    stroke_width_submenu.add_radiobutton(label="5",
                                  variable=self.stroke_size,
                                  value=5)
    color_menu.add_cascade(label="Stroke Size",
                    menu=stroke_width_submenu)
    the_menu.add_cascade(label="Color", menu=color_menu)

    # Display the menu bar
    root.config(menu=the_menu)

# ---------- CATCH MOUSE UP ----------

def left_but_down(self, event=None):
    self.left_but = "down"

    # Set x & y when mouse is clicked
    self.x1_line_pt = event.x
    self.y1_line_pt = event.y

# ---------- CATCH MOUSE UP ----------
```

```
    def left_but_up(self, event=None):
        self.left_but = "up"

        # Reset the line
        self.x_pos = None
        self.y_pos = None

        # Set x & y when mouse is released
        self.x2_line_pt = event.x
        self.y2_line_pt = event.y

        # If mouse is released and line tool is selected
        # draw the line
        if self.drawing_tool.get() == "line":
            self.line_draw(event)
        elif self.drawing_tool.get() == "arc":
            self.arc_draw(event)
        elif self.drawing_tool.get() == "oval":
            self.oval_draw(event)
        elif self.drawing_tool.get() == "rectangle":
            self.rectangle_draw(event)
        elif self.drawing_tool.get() == "text":
            self.text_draw(event)

    # ---------- CATCH MOUSE MOVEMENT ----------

    def motion(self, event=None):
        if self.drawing_tool.get() == "pencil":
            self.pencil_draw(event)

    # ---------- DRAW PENCIL ----------

    def pencil_draw(self, event=None):
        if self.left_but == "down":

            # Make sure x and y have a value
            if self.x_pos is not None and self.y_pos is not None:
                event.widget.create_line(self.x_pos, self.y_pos, event.x, event.y, smooth=TRUE,
fill=self.stroke_color.get(), width=self.stroke_size.get())

            self.x_pos = event.x
            self.y_pos = event.y

    # ---------- NEW DRAWING METHODS ----------

    def line_draw(self, event=None):

        # Shortcut way to check if none of these values contain None
        if None not in (self.x1_line_pt, self.y1_line_pt, self.x2_line_pt, self.y2_line_pt):
            event.widget.create_line(self.x1_line_pt, self.y1_line_pt, self.x2_line_pt, self.y2_line_pt,
smooth=TRUE, fill=self.stroke_color.get())

    def arc_draw(self, event=None):
```

```
    # Shortcut way to check if none of these values contain None
    if None not in (self.x1_line_pt, self.y1_line_pt, self.x2_line_pt, self.y2_line_pt):
        coords = self.x1_line_pt, self.y1_line_pt, self.x2_line_pt, self.y2_line_pt

        # start : starting angle for the slice in degrees
        # extent : width of the slice in degrees
        # fill : fill color if needed
        # style : can be ARC, PIESLICE, or CHORD
        event.widget.create_arc(coords, start=0, extent=150,
                                style=ARC, fill=self.stroke_color.get())

def oval_draw(self, event=None):
    if None not in (self.x1_line_pt, self.y1_line_pt, self.x2_line_pt, self.y2_line_pt):
        # fill : Color option names are here http://wiki.tcl.tk/37701
        # outline : border color
        # width : width of border in pixels

        event.widget.create_oval(self.x1_line_pt, self.y1_line_pt, self.x2_line_pt, self.y2_line_pt,
                                 fill=self.fill_color.get(),
                                 outline=self.stroke_color.get(),
                                 width=self.stroke_size.get())

def rectangle_draw(self, event=None):
    if None not in (self.x1_line_pt, self.y1_line_pt, self.x2_line_pt, self.y2_line_pt):
        # fill : Color option names are here http://wiki.tcl.tk/37701
        # outline : border color
        # width : width of border in pixels

        event.widget.create_rectangle(self.x1_line_pt, self.y1_line_pt,
                                      self.x2_line_pt, self.y2_line_pt,
                                      fill=self.fill_color.get(),
                                      outline=self.stroke_color.get(),
                                      width=self.stroke_size.get())

def text_draw(self, event=None):
    if None not in (self.x1_line_pt, self.y1_line_pt):
        # Show all fonts available
        # print(tkinter.font.families())

        text_font = tkinter.font.Font(family=self.text_font.get(),
                                      size=self.text_size.get(), weight=self.bold_text.get(),
                                      slant=self.italic_text.get())
        # Get the text the user wants to enter
        user_text = simpledialog.askstring("Input",
                                           "Enter Text", parent=root)
        if user_text is not None:
            event.widget.create_text(self.x1_line_pt, self.y1_line_pt,
                                     fill=self.fill_color.get(),
                                     font=text_font,
                                     text=user_text)

# ---------- END OF NEW DRAWING METHODS ----------
```

```
    def pick_fill(self, event=None):
        fill_color = askcolor(title='Pick Fill color')
        if None not in fill_color:
            self.fill_color.set(fill_color[1])
            print("Color ", self.fill_color.get())

    def pick_stroke(self, event=None):
        stroke_color = askcolor(title='Pick Stroke color')
        if None not in stroke_color:
            self.stroke_color.set(stroke_color[1])

    def __init__(self, root):
        drawing_area = Canvas(root, width=800, height=600)
        drawing_area.pack()

        self.text_font.set("Times")
        self.text_size.set(20)

        # NEW FIX DEFAULTS
        self.bold_text.set('normal')
        self.italic_text.set('roman')

        self.drawing_tool.set("pencil")
        self.stroke_size.set(3)
        self.fill_color.set('#000000')
        self.stroke_color.set('#000000')

        self.make_menu_bar()

        # Set focus for catching events to the canvas
        drawing_area.focus_force()
        drawing_area.bind("<Motion>", self.motion)
        drawing_area.bind("<ButtonPress-1>", self.left_but_down)
        drawing_area.bind("<ButtonRelease-1>", self.left_but_up)


paint_app = PaintApp(root)
root.mainloop()
```

 # ---------- TKINTER PAINT APP 4 ----------

```
from tkinter import *
import tkinter.font
from tkinter.colorchooser import *
from tkinter import simpledialog

# NEW Used to save image data and fonts
from PIL import Image, ImageDraw, ImageTk

# NEW Used to get the file name for saving
import os

# NEW The save dialog
```

```
import tkinter.filedialog

# Create main window
root = Tk()
root.geometry("800x600")


class PaintApp:
    text_font = StringVar()
    text_size = IntVar()
    bold_text = StringVar()
    italic_text = StringVar()

    # Stores current tool we are using
    drawing_tool = StringVar()

    # STORE DRAWING SETTINGS
    stroke_size = IntVar()
    fill_color = StringVar()
    stroke_color = StringVar()

    # Tracks whether left mouse is down
    left_but = "up"

    # x and y positions for drawing with pencil
    x_pos, y_pos = None, None

    # Tracks x & y when the mouse is clicked and released
    x1_line_pt, y1_line_pt, x2_line_pt, y2_line_pt = None, None, None, None

    # NEW Empty image to draw on with width, height and color
    my_image = Image.new("RGB", (800, 600), (255, 255, 255))

    # NEW Used to draw shapes
    draw = ImageDraw.Draw(my_image)

    # NEW created so I can draw files to canvas
    drawing_area = Canvas(root, width=800, height=600)

    # Quits the TkInter app when called
    @staticmethod
    def quit_app():
        root.quit()

    # New saves PIL image as PNG
    def save_file(self, event=None):
        # Opens the save as dialog box
        file = tkinter.filedialog.asksaveasfile(mode='w', defaultextension=".png")
        if file:
            file_path = os.path.abspath(file.name)
            self.my_image.save(file_path)

    # NEW Opens the PIL image
    def open_file(self, event=None):
```

```
    file_path = tkinter.filedialog.askopenfilename(parent=root)
    if file_path:

        # Load the image
        my_pic = Image.open(file_path)

        # Put the image in a canvas class
        self.drawing_area.image = ImageTk.PhotoImage(my_pic)

        # Draw the image starting in the upper left
        self.drawing_area.create_image(0, 0,
                        image=self.drawing_area.image,
                        anchor='nw')


def make_menu_bar(self):
    # Create the menu object
    the_menu = Menu(root)

    # ---- FILE MENU ----
    # Create a pull down menu that can't be removed
    file_menu = Menu(the_menu, tearoff=0)

    # Add items to the menu that show when clicked
    # compound allows you to add an image

    # NEW Add option to open and save files
    file_menu.add_command(label="Open",
                        command=self.open_file)
    file_menu.add_command(label="Save",
                        command=self.save_file)

    # Add a horizontal bar to group similar commands
    file_menu.add_separator()

    # Call for the function to execute when clicked
    file_menu.add_command(label="Quit", command=self.quit_app)

    # Add the pull down menu to the menu bar
    the_menu.add_cascade(label="File", menu=file_menu)

    # ---- FONT MENU ----
    font_menu = Menu(the_menu, tearoff=0)
    font_type_submenu = Menu(font_menu)
    font_type_submenu.add_radiobutton(label="Times",
                            variable=self.text_font)
    font_type_submenu.add_radiobutton(label="Courier",
                            variable=self.text_font)
    font_type_submenu.add_radiobutton(label="Ariel",
                            variable=self.text_font)
    font_menu.add_cascade(label="Font Type",
                        menu=font_type_submenu)

    font_size_submenu = Menu(font_menu)
```

```
font_size_submenu.add_radiobutton(label="10",
                                  variable=self.text_size,
                                  value=10)
font_size_submenu.add_radiobutton(label="15",
                                  variable=self.text_size,
                                  value=15)
font_size_submenu.add_radiobutton(label="20",
                                  variable=self.text_size,
                                  value=20)
font_size_submenu.add_radiobutton(label="25",
                                  variable=self.text_size,
                                  value=25)
font_menu.add_cascade(label="Font Size",
                      menu=font_size_submenu)

# NEW FIX THE ON AND OFF VALUES FOR BOLD & ITALIC
font_menu.add_checkbutton(label="Bold",
                          variable=self.bold_text,
                          onvalue='bold',
                          offvalue='normal')
font_menu.add_checkbutton(label="Italic",
                          variable=self.italic_text,
                          onvalue='italic',
                          offvalue='roman')

the_menu.add_cascade(label="Font", menu=font_menu)

# ---- TOOL MENU ----
tool_menu = Menu(the_menu, tearoff=0)
tool_menu.add_radiobutton(label="Pencil",
                          variable=self.drawing_tool,
                          value="pencil")
tool_menu.add_radiobutton(label="Line",
                          variable=self.drawing_tool,
                          value="line")
tool_menu.add_radiobutton(label="Arc",
                          variable=self.drawing_tool,
                          value="arc")
tool_menu.add_radiobutton(label="Oval",
                          variable=self.drawing_tool,
                          value="oval")
tool_menu.add_radiobutton(label="Rectangle",
                          variable=self.drawing_tool,
                          value="rectangle")
tool_menu.add_radiobutton(label="Text",
                          variable=self.drawing_tool,
                          value="text")
the_menu.add_cascade(label="Tool", menu=tool_menu)

# ---- COLOR MENU ----

color_menu = Menu(the_menu, tearoff=0)
color_menu.add_command(label="Fill", command=self.pick_fill)
color_menu.add_command(label="Stroke", command=self.pick_stroke)
```

```
    stroke_width_submenu = Menu(color_menu)
    stroke_width_submenu.add_radiobutton(label="2",
                                variable=self.stroke_size,
                                value=2)
    stroke_width_submenu.add_radiobutton(label="3",
                                  variable=self.stroke_size,
                                  value=3)
    stroke_width_submenu.add_radiobutton(label="4",
                                  variable=self.stroke_size,
                                  value=4)
    stroke_width_submenu.add_radiobutton(label="5",
                                  variable=self.stroke_size,
                                  value=5)
    color_menu.add_cascade(label="Stroke Size",
                           menu=stroke_width_submenu)
    the_menu.add_cascade(label="Color", menu=color_menu)

    # Display the menu bar
    root.config(menu=the_menu)

# ---------- CATCH MOUSE UP ----------

def left_but_down(self, event=None):
    self.left_but = "down"

    # Set x & y when mouse is clicked
    self.x1_line_pt = event.x
    self.y1_line_pt = event.y

# ---------- CATCH MOUSE UP ----------

def left_but_up(self, event=None):
    self.left_but = "up"

    # Reset the line
    self.x_pos = None
    self.y_pos = None

    # Set x & y when mouse is released
    self.x2_line_pt = event.x
    self.y2_line_pt = event.y

    # If mouse is released and line tool is selected
    # draw the line
    if self.drawing_tool.get() == "line":
        self.line_draw(event)
    elif self.drawing_tool.get() == "arc":
        self.arc_draw(event)
    elif self.drawing_tool.get() == "oval":
        self.oval_draw(event)
    elif self.drawing_tool.get() == "rectangle":
        self.rectangle_draw(event)
    elif self.drawing_tool.get() == "text":
        self.text_draw(event)
```

```
    # ---------- CATCH MOUSE MOVEMENT ----------

    def motion(self, event=None):
        if self.drawing_tool.get() == "pencil":
            self.pencil_draw(event)

    # ---------- DRAW PENCIL ----------

    def pencil_draw(self, event=None):
        if self.left_but == "down":

            # Make sure x and y have a value
            if self.x_pos is not None and self.y_pos is not None:
                event.widget.create_line(self.x_pos, self.y_pos, event.x, event.y, smooth=TRUE,
fill=self.stroke_color.get(), width=self.stroke_size.get())

                # NEW Draw to PIL image for saving
                self.draw.line([(self.x_pos, self.y_pos),
                                (event.x, event.y)],
                               fill=self.stroke_color.get(),
                               width=self.stroke_size.get())

            self.x_pos = event.x
            self.y_pos = event.y

    def line_draw(self, event=None):

        # Shortcut way to check if none of these values contain None
        if None not in (self.x1_line_pt, self.y1_line_pt, self.x2_line_pt, self.y2_line_pt):
            event.widget.create_line(self.x1_line_pt, self.y1_line_pt, self.x2_line_pt, self.y2_line_pt,
smooth=TRUE, fill=self.stroke_color.get())

            # NEW Draw to PIL image for saving
            self.draw.line([(self.x1_line_pt, self.y1_line_pt),
                            (self.x2_line_pt, self.y2_line_pt)],
                           fill=self.stroke_color.get(),
                           width=self.stroke_size.get())

    def arc_draw(self, event=None):

        # Shortcut way to check if none of these values contain None
        if None not in (self.x1_line_pt, self.y1_line_pt, self.x2_line_pt, self.y2_line_pt):
            coords = self.x1_line_pt, self.y1_line_pt, self.x2_line_pt, self.y2_line_pt

            event.widget.create_arc(coords, start=0, extent=150,
                                    style=ARC, fill=self.stroke_color.get())

            # NEW Draw to PIL image for saving
            self.draw.arc([(self.x1_line_pt, self.y1_line_pt),
                           (self.x2_line_pt, self.y2_line_pt)],
                          start=0, end=150,
                           fill=self.stroke_color.get())
```

```
def oval_draw(self, event=None):
    if None not in (self.x1_line_pt, self.y1_line_pt, self.x2_line_pt, self.y2_line_pt):
        # fill : Color option names are here http://wiki.tcl.tk/37701
        # outline : border color
        # width : width of border in pixels

        event.widget.create_oval(self.x1_line_pt, self.y1_line_pt, self.x2_line_pt, self.y2_line_pt,
                                 fill=self.fill_color.get(),
                                 outline=self.stroke_color.get(),
                                 width=self.stroke_size.get())

        # NEW Draw oval to PIL image
        self.draw.ellipse([(self.x1_line_pt, self.y1_line_pt),
                           (self.x2_line_pt, self.y2_line_pt)],
                          fill=self.fill_color.get(),
                          outline=self.stroke_color.get())

def rectangle_draw(self, event=None):
    if None not in (self.x1_line_pt, self.y1_line_pt, self.x2_line_pt, self.y2_line_pt):
        # fill : Color option names are here http://wiki.tcl.tk/37701
        # outline : border color
        # width : width of border in pixels

        event.widget.create_rectangle(self.x1_line_pt, self.y1_line_pt,
                                      self.x2_line_pt, self.y2_line_pt,
                                      fill=self.fill_color.get(),
                                      outline=self.stroke_color.get(),
                                      width=self.stroke_size.get())

        # NEW Draw rectangle to PIL image
        self.draw.rectangle([(self.x1_line_pt, self.y1_line_pt),
                             (self.x2_line_pt, self.y2_line_pt),
                             ], fill=self.fill_color.get(),
                                      outline=self.stroke_color.get())

def text_draw(self, event=None):
    if None not in (self.x1_line_pt, self.y1_line_pt):
        # Show all fonts available
        # print(tkinter.font.families())

        text_font = tkinter.font.Font(family=self.text_font.get(),
                                      size=self.text_size.get(), weight=self.bold_text.get(),
                                      slant=self.italic_text.get())
        # Get the text the user wants to enter
        user_text = simpledialog.askstring("Input",
                                           "Enter Text", parent=root)
        if user_text is not None:
            event.widget.create_text(self.x1_line_pt, self.y1_line_pt,
                                     fill=self.fill_color.get(),
                                     font=text_font,
                                     text=user_text)

            # NEW While you can save text to a PIL file you have
            # to put either True Type Fonts, or PIL fonts in the
```

```
            # directory or provide specific paths based on your
            # computer. You could also convert from fonts you have
            # to PIL fonts. This is beyond this tutorial so I leave
            # that task to you for homework

    def pick_fill(self, event=None):
        fill_color = askcolor(title='Pick Fill color')
        if None not in fill_color:
            self.fill_color.set(fill_color[1])
            print("Color ", self.fill_color.get())

    def pick_stroke(self, event=None):
        stroke_color = askcolor(title='Pick Stroke color')
        if None not in stroke_color:
            self.stroke_color.set(stroke_color[1])

    def __init__(self, root):

        self.drawing_area.pack()

        self.text_font.set("Times")
        self.text_size.set(20)
        self.bold_text.set('normal')
        self.italic_text.set('roman')

        self.drawing_tool.set("pencil")
        self.stroke_size.set(3)
        self.fill_color.set('#000000')
        self.stroke_color.set('#000000')

        self.make_menu_bar()

        # Set focus for catching events to the canvas
        self.drawing_area.focus_force()
        self.drawing_area.bind("<Motion>", self.motion)
        self.drawing_area.bind("<ButtonPress-1>", self.left_but_down)
        self.drawing_area.bind("<ButtonRelease-1>", self.left_but_up)


paint_app = PaintApp(root)
root.mainloop()

#---------- INSERT_DATA.PY ----------

import mysql.connector
from mysql.connector import Error
from mysql.connector import errorcode
from datetime import datetime

try:
    # Create a connection with the database
    conn = mysql.connector.connect(host='localhost',
    database='test1', user='studentadmin',
    password='TurtleDove')
```

```
    # Query used to insert data
    # query = "INSERT INTO students VALUES('Dale', 'Cooper', 'dcooper@aol.com','123 Main
St', 'Yakima', 'WA', 98901, '792-223-8901', '1959-2-22','M', NOW(), 3.50, NULL)"

    # 2. Create a parameterized query
    # query = "INSERT INTO students (first_name, last_name, email, street, city, state, zip,
phone, birth_date, sex, date_entered, lunch_cost, student_id) VALUES (%s, %s, %s, %s, %s,
%s, %s, %s, %s, %s, %s, %s, %s)"

    # 2. Get the current time and format it to fit what
    # MySQL expects
    # now_time = datetime.now()
    # format_date = now_time.strftime('%Y-%m-%d %H:%M:%S')

    # 2. Insert multiple rows
    # You must use None instead of NULL
    # students = [('Harry', 'Truman', 'htruman@aol.com', '202 South St', 'Vancouver', 'WA',
98660, '792-223-9810', '1946-1-24','M', format_date, 3.50, None),
    # ('Shelly', 'Johnson', 'sjohnson@aol.com', '9 Pond Rd', 'Sparks', 'NV', 89431,
'792-223-6734', '1970-12-12','F', format_date, 3.50, None),
    # ('Bobby', 'Briggs', 'bbriggs@aol.com', '14 12th St', 'San Diego', 'CA',92101,
'792-223-6178', '1967-5-24','M', format_date, 3.50, None),
    # ('Donna', 'Hayward', 'dhayward@aol.com', '120 16th St', 'Davenport', 'IA', 52801,
'792-223-2001', '1970-3-24','F', format_date, 3.50, None),
    # ('Audrey', 'Horne', 'ahorne@aol.com', '342 19th St', 'Detroit', 'MI', 48222, '792-223-2001',
'1965-2-1','F', format_date, 3.50, None),
    # ('James', 'Hurley', 'jhurley@aol.com', '2578 Cliff St', 'Queens', 'NY', 11427,
'792-223-1890', '1967-1-2','M', format_date, 3.50, None),
    # ('Lucy', 'Moran', 'lmoran@aol.com', '178 Dover St', 'Hollywood', 'CA', 90078,
'792-223-9678', '1954-11-27','F', format_date, 3.50, None),
    # ('Tommy', 'Hill', 'thill@aol.com', '672 High Plains', 'Tucson', 'AZ', 85701, '792-223-1115',
'1951-12-21','M', format_date, 3.50, None),
    # ('Andy', 'Brennan', 'abrennan@aol.com', '281 4th St', 'Jacksonville', 'NC', 28540,
'792-223-8902', '1960-12-27','M', format_date, 3.50, None)]

    # 3. Insert multiple rows with one query
    # query = "INSERT INTO classes VALUES ('English', NULL), ('Speech', NULL), ('Literature',
NULL),('Algebra', NULL), ('Geometry', NULL), ('Trigonometry', NULL), ('Calculus', NULL), ('Earth
Science', NULL), ('Biology', NULL), ('Chemistry', NULL), ('Physics', NULL), ('History', NULL),
('Art', NULL), ('Gym', NULL)";

    # 4. Enter test data
    # query = "INSERT INTO tests VALUES ('2014-8-25', 'Q', 15, 1, NULL), ('2014-8-27', 'Q', 15,
1, NULL), ('2014-8-29', 'T', 30, 1, NULL), ('2014-8-29', 'T', 30, 2, NULL), ('2014-8-27', 'Q', 15,
4, NULL), ('2014-8-29', 'T', 30, 4, NULL)"

    # 5. Insert score data
    # query = "INSERT INTO scores VALUES (1, 1, 15),(1, 2, 14),(1, 3, 28),(1, 4, 29),(1, 5, 15),(1, 6,
27),(2, 1, 15),(2, 2, 14),(2, 3, 26),(2, 4, 28),(2, 5, 14),(2, 6, 26),(3, 1, 14),(3, 2, 14),(3, 3, 26),(3, 4,
26),(3, 5, 13),(3, 6, 26),(4, 1, 15),(4, 2, 14),(4, 3, 27),(4, 4, 27),(4, 5, 15),(4, 6, 27),(5, 1, 14),(5, 2,
13),(5, 3, 26),(5, 4, 27),(5, 5, 13),(5, 6, 27),(6, 1, 13),(6, 2, 13),(6, 4, 26),(6, 5, 13),(6, 6, 26),(7, 1,
13),(7, 2, 13),(7, 3, 25),(7, 4, 27),(7, 5, 13),(8, 1, 14),(8, 3, 26),(8, 4, 23),(8, 5, 12),(8, 6, 24),(9, 1,
```

```
15),(9, 2, 13),(9, 3, 28),(9, 4, 27),(9, 5, 14),(9, 6, 27),(10, 1, 15),(10, 2, 13),(10, 3, 26),(10, 4, 27),
(10, 5, 12),(10, 6, 22)"

    # 6. Insert absences
    query = "INSERT INTO absences VALUES (6, '2014-08-29'),(7, '2014-08-29'),(8,
'2014-08-27')"

    # The cursor object provides methods we can use to
    # interact with the database
    cursor = conn.cursor()
    # Execute the query
    # cursor.execute(query)

    # 2. Insert multiple rows of data from the list
    # cursor.executemany(query, students)

    # 3. Insert multiple rows with one query
    cursor.execute(query)

    # Send the transaction to MySQL
    conn.commit()
    print("Data Entered")
    # Reset results and close the cursor
    cursor.close()

# Catch any errors
except mysql.connector.Error as error:
    print("Error :", error)

# Always executes and makes sure the DB connection is
# released
finally:
    if(conn.is_connected()):
        conn.close()
        print("Database Connection Closed")

# ---------- SELECT_DATA.PY ----------

import mysql.connector
from mysql.connector import Error

try:
    # Create a connection with the database
    conn = mysql.connector.connect(host='localhost',
    database='students', user='studentadmin',
    password='TurtleDove')

    # Get a list of all students
    query = "SELECT * FROM students"

    # 2. Get 1st and last from the state of WA
    query = "SELECT first_name, last_name FROM students WHERE state='WA'"

    # 3. Get students born after 1965
```

```
    # You can compare values with =, >, <, >=, <=, !=
    # To get the month, day or year of a date use MONTH(),
    # DAY(), or YEAR()
    query = "SELECT first_name, last_name FROM students WHERE YEAR(birth_date) >= 1965"

    # 4. Use or to use multiple conditions
    # AND, && : Returns a true value if both conditions are true
    # OR, || : Returns a true value if either condition is true
    # NOT, ! : Returns a true value if the operand is false
    query = 'SELECT first_name, last_name, birth_date FROM students WHERE MONTH(birth_-
date) = 2 OR state="CA"'

    # 5. Double up logical operators
    query = 'SELECT last_name, state, birth_date FROM students WHERE DAY(birth_date) >=
12 && (state="CA" || state="NV")'

    # 6. Check for NULL with IS NULL or IS NOT NULL
    query = 'SELECT first_name, last_name FROM students WHERE last_name IS NULL'

    # 7. Use ORDER BY to alphabetize data
    # To change the order use ORDER BY col_name DESC;
    # Limit defines how many results you want
    # LIMIT 5, 10 returns the 5th through 10th results
    query = 'SELECT first_name, last_name FROM students ORDER BY last_name LIMIT 5'

    # 8 Use CONCAT to join columns and AS to create
    # aliases
    query = 'SELECT CONCAT(first_name, " ", last_name) AS "Name", CONCAT(city, ", ", state)
AS "Hometown" FROM students'

    # 9. Use LIKE to find data that meets limited definitions
    # Matches first name that starts with D or last name
    # that ends with n
    # % matches any series of characters
    query = 'SELECT last_name, first_name FROM students WHERE first_name LIKE "D%" OR
last_name LIKE "%n"'

    # 10. _ is used with LIKE to match any character
    # Find 4 letters followed by a y for a 1st name
    query = 'SELECT last_name, first_name FROM students WHERE first_name LIKE "____y"'

    # 11. Get the number of boys and girls with COUNT
    # GROUP BY defines how the results will be grouped
    query = 'SELECT sex, COUNT(*) FROM students GROUP BY sex'

    # 12. Find the number of birthdays in each month
    query = 'SELECT MONTH(birth_date) AS "Month", COUNT(*) FROM students GROUP BY
Month ORDER BY Month'

    # 13. Only receive results if a state has more then
    # 1 student with HAVING
    query = 'SELECT state, COUNT(state) AS "Amount" FROM students GROUP BY state HAV-
ING Amount > 1'
```

```
    # 14. Use DISTINCT to only receive a result once
    # Get states in which students were born
    query = 'SELECT DISTINCT state FROM students ORDER BY state'

    # 15. Get the number of states from which stuents were born
    query = 'SELECT COUNT(DISTINCT state) FROM students'

    cursor = conn.cursor()
    cursor.execute(query)
    students = cursor.fetchall()
    print("Total Results :", len(students))

    # Get the first and last name using indexes
    # for s in students:
    #     print(s[1], " ", s[2])

    # 2 - 13. Get 2 results
    # for s in students:
    #     print(s[0], " ", s[1])

    # 14 - 15. Get 1 Result
    for s in students:
        print(s[0])

# Catch any errors
except mysql.connector.Error as error:
    print("Error :", error)

# Always executes and makes sure the DB connection is
# released
finally:
    if(conn.is_connected()):
        conn.close()
        print("Database Connection Closed")


---------- SELECT_DATA3.PY ----------

import mysql.connector
from mysql.connector import Error

try:
    conn = mysql.connector.connect(host='localhost', database='test1', user='studentadmin',
password='TurtleDove')

    cursor = conn.cursor()

    # 1. Get test data
    # query = 'SELECT test_id, MIN(score), MAX(score), MAX(score) - MIN(score), SUM(score),
AVG(score) FROM scores GROUP BY test_id'
    # cursor.execute(query)
    # results = cursor.fetchall()

    # 2. Find out how many tests student 6 took
```

```
    # query = 'SELECT student_id, test_id FROM scores WHERE student_id=6'
    # cursor.execute(query)
    # results = cursor.fetchall()

    # 3. Insert a test make up, delete student from absence
    # query = 'INSERT INTO scores VALUES (6, 3, 24)'
    # cursor.execute(query)
    # query = 'DELETE FROM absences WHERE student_id = 6'
    # cursor.execute(query)
    # query = 'SELECT student_id, test_id FROM scores WHERE student_id=6'
    # cursor.execute(query)
    # results = cursor.fetchall()

    # 4. You can alter tables
    # Add a test taken column
    # query = 'ALTER TABLE absences ADD COLUMN test_taken CHAR(1) NOT NULL DEFAULT
"F" AFTER student_id'
    # cursor.execute(query)
    # Change the data type for test_taken
    # query = 'ALTER TABLE absences MODIFY COLUMN test_taken ENUM("T","F") NOT NULL
DEFAULT "F"'
    # cursor.execute(query)

    # 5. You can delete columns
    # query = 'ALTER TABLE absences DROP COLUMN test_taken'
    # cursor.execute(query)

    # 6. Use update to change a value in a row
    # query = 'UPDATE scores SET score=25 WHERE student_id=4 AND test_id=3'
    # cursor.execute(query)

    # 7. Use BETWEEN to find matches in a range
    # query = 'SELECT first_name, last_name, birth_date FROM students WHERE birth_date
BETWEEN "1960-1-1" AND "1970-1-1"'
    # cursor.execute(query)
    # results = cursor.fetchall()

    # 8. Use IN to narrow results based on a list
    # query = 'SELECT first_name, last_name, student_id FROM students WHERE first_name IN
("Bobby", "Lucy", "Andy")'
    # cursor.execute(query)
    # results = cursor.fetchall()

    # 9. Use JOIN to combine data from multiple tables
    # You have to define the 2 tables to join after FROM
    # You have to define the common data between the tables after WHERE
    # It is good to qualify the specific data needed by proceeding
    # it with the tables name and a period
    # query = 'SELECT scores.student_id, tests.date, scores.score, tests.maxscore FROM tests,
scores WHERE date = "2014-08-25" AND tests.test_id = scores.test_id'
    # cursor.execute(query)
    # results = cursor.fetchall()

    # 10. You can JOIN more then 2 tables as long as you define the like
```

```
    # data between those tables
    # query = 'SELECT CONCAT(students.first_name, " ", students.last_name) AS Name, tests.-
date, scores.score, tests.maxscore FROM tests, scores, students WHERE date = "2014-08-25"
AND tests.test_id = scores.test_id AND scores.student_id = students.student_id'
    # cursor.execute(query)
    # results = cursor.fetchall()

    # 11. If we wanted a list of the number of absences per student we
    # have to group by student_id or we would get just one result
    # query = 'SELECT students.student_id, students.first_name, students.last_name,
COUNT(absences.date) FROM students, absences WHERE students.student_id = absences.s-
tudent_id GROUP BY students.student_id'
    # cursor.execute(query)
    # results = cursor.fetchall()

    # 12. An INNER JOIN gets all rows of data from both tables if there is a
    # match between columns in both tables
    query = 'SELECT students.first_name, students.last_name, scores.test_id, scores.score
FROM students INNER JOIN scores ON students.student_id=scores.student_id WHERE
scores.score <= 15 ORDER BY scores.test_id'
    cursor.execute(query)
    results = cursor.fetchall()

    # 1. Get test score data
    # for x in results:
    #     print(x[0], " Min :", x[1], " Max :", x[2], " Rng :", x[3], " Sum :", x[4], " Avg :", x[5])

    # 2 - 3. Get 2 results
    # for x in results:
    #     print(x[0], " ", x[1])

    # 7 - 8. 3 Outputs
    # for x in results:
    #     print(x[0], " ", x[1], " ", x[2])

    # 9 - 12 : 4 Outputs
    for x in results:
        print(x[0], " ", x[1], " ", x[2], " ", x[3])

except mysql.connector.Error as error:
    print("Error :", error)
finally:
    if(conn.is_connected()):
        conn.close()


---------- STUDENT_DB.PY ----------

from tkinter import *
from tkinter import ttk

class StudentDB:
    # Used as te headers for the treeview table
```

```
    headers = ['ID', 'First Name', 'Last Name', 'Email', 'Street', 'City', 'State', 'Zip', 'Phone',
'Birth', 'Sex', 'Lunch']
    # Sample data used to test the look of the treeview
    student_info = [
    (1, 'Dale', 'Cooper', 'dcooper@aol.com','123 Main St', 'Yakima', 'WA', 98901,
'792-223-8901', '1959-2-22', 'M', 3.50),
    (2, 'Harry', 'Truman', 'htruman@aol.com', '202 South St', 'Vancouver', 'WA', 98660,
'792-223-9810', '1946-1-24','M', 3.50),
    (3, 'Shelly', 'Johnson', 'sjohnson@aol.com', '9 Pond Rd', 'Sparks', 'NV', 89431,
'792-223-6734', '1970-12-12','F', 3.50),
    (3, 'Shelly', 'Johnson', 'sjohnson@aol.com', '9 Pond Rd', 'Sparks', 'NV', 89431,
'792-223-6734', '1970-12-12','F', 3.50),
    (3, 'Shelly', 'Johnson', 'sjohnson@aol.com', '9 Pond Rd', 'Sparks', 'NV', 89431,
'792-223-6734', '1970-12-12','F', 3.50),
    (3, 'Shelly', 'Johnson', 'sjohnson@aol.com', '9 Pond Rd', 'Sparks', 'NV', 89431,
'792-223-6734', '1970-12-12','F', 3.50),
    (3, 'Shelly', 'Johnson', 'sjohnson@aol.com', '9 Pond Rd', 'Sparks', 'NV', 89431,
'792-223-6734', '1970-12-12','F', 3.50),
    (3, 'Shelly', 'Johnson', 'sjohnson@aol.com', '9 Pond Rd', 'Sparks', 'NV', 89431,
'792-223-6734', '1970-12-12','F', 3.50),
    (3, 'Shelly', 'Johnson', 'sjohnson@aol.com', '9 Pond Rd', 'Sparks', 'NV', 89431,
'792-223-6734', '1970-12-12','F', 3.50),
    (3, 'Shelly', 'Johnson', 'sjohnson@aol.com', '9 Pond Rd', 'Sparks', 'NV', 89431,
'792-223-6734', '1970-12-12','F', 3.50),
    (3, 'Shelly', 'Johnson', 'sjohnson@aol.com', '9 Pond Rd', 'Sparks', 'NV', 89431,
'792-223-6734', '1970-12-12','F', 3.50),
    (3, 'Shelly', 'Johnson', 'sjohnson@aol.com', '9 Pond Rd', 'Sparks', 'NV', 89431,
'792-223-6734', '1970-12-12','F', 3.50),
    (3, 'Shelly', 'Johnson', 'sjohnson@aol.com', '9 Pond Rd', 'Sparks', 'NV', 89431,
'792-223-6734', '1970-12-12','F', 3.50),
    (3, 'Shelly', 'Johnson', 'sjohnson@aol.com', '9 Pond Rd', 'Sparks', 'NV', 89431,
'792-223-6734', '1970-12-12','F', 3.50),
    (3, 'Shelly', 'Johnson', 'sjohnson@aol.com', '9 Pond Rd', 'Sparks', 'NV', 89431,
'792-223-6734', '1970-12-12','F', 3.50),
    (3, 'Shelly', 'Johnson', 'sjohnson@aol.com', '9 Pond Rd', 'Sparks', 'NV', 89431,
'792-223-6734', '1970-12-12','F', 3.50),
    (3, 'Shelly', 'Johnson', 'sjohnson@aol.com', '9 Pond Rd', 'Sparks', 'NV', 89431,
'792-223-6734', '1970-12-12','F', 3.50)
    ]

    def __init__(self):
        # Will hold all the student data in a treeview table
        self.tree = None
        self.create_widgets()

    # Initializes all of the widgets in our app
    def create_widgets(self):
        # ----- ROW 1 -----
        # Create the lable and place it in the upper left hand corner using
        # the grid layout
        sid_label = Label(root, text='ID')
        sid_label.grid(row=0, column=0, padx=5, pady=10, sticky=W)

        # Will hold the values entered into the entry widget
```

```
self.sid_entry_value = StringVar(root, value="")

# Create the entry widget and assign all values entered
# into it to the StringVar
self.sid_entry = ttk.Entry(root,
                    textvariable=self.sid_entry_value)
self.sid_entry.grid(row=0, column=1, padx=5, pady=10, sticky=W)

f_name_label = Label(root, text='First Name')
f_name_label.grid(row=0, column=2, padx=5, pady=10, sticky=W)
self.f_name_entry_value = StringVar(root, value="")
self.f_name_entry = ttk.Entry(root,
                    textvariable=self.f_name_entry_value)
self.f_name_entry.grid(row=0, column=3, padx=5, pady=10, sticky=W)

l_name_label = Label(root, text='Last Name')
l_name_label.grid(row=0, column=4, padx=5, pady=10, sticky=W)
self.l_name_entry_value = StringVar(root, value="")
self.l_name_entry = ttk.Entry(root,
                    textvariable=self.l_name_entry_value)
self.l_name_entry.grid(row=0, column=5, padx=5, pady=10, sticky=W)

email_label = Label(root, text='Email')
email_label.grid(row=0, column=6, padx=5, pady=10, sticky=W)
self.email_entry_value = StringVar(root, value="")
self.email_entry = ttk.Entry(root,
                    textvariable=self.email_entry_value)
self.email_entry.grid(row=0, column=7, padx=5, pady=10, sticky=W)

street_label = Label(root, text='Street')
street_label.grid(row=0, column=8, padx=5, pady=10, sticky=W)
self.street_entry_value = StringVar(root, value="")
self.street_entry = ttk.Entry(root,
                    textvariable=self.street_entry_value)
self.street_entry.grid(row=0, column=9, padx=5, pady=10, sticky=W)

# ----- 2nd ROW -----
city_label = Label(root, text='City')
city_label.grid(row=1, column=0, padx=5, pady=10, sticky=W)
self.city_entry_value = StringVar(root, value="")
self.city_entry = ttk.Entry(root,
                    textvariable=self.city_entry_value)
self.city_entry.grid(row=1, column=1, padx=5, pady=10, sticky=W)

state_label = Label(root, text='State')
state_label.grid(row=1, column=2, padx=5, pady=10, sticky=W)
self.state_entry_value = StringVar(root, value="")
self.state_entry = ttk.Entry(root,
                    textvariable=self.state_entry_value)
self.state_entry.grid(row=1, column=3, padx=5, pady=10, sticky=W)

zip_label = Label(root, text='Zip Code')
zip_label.grid(row=1, column=4, padx=5, pady=10, sticky=W)
self.zip_entry_value = StringVar(root, value="")
```

```
        self.zip_entry = ttk.Entry(root,
                                   textvariable=self.zip_entry_value)
        self.zip_entry.grid(row=1, column=5, padx=5, pady=10, sticky=W)

        phone_label = Label(root, text='Phone')
        phone_label.grid(row=1, column=6, padx=5, pady=10, sticky=W)
        self.phone_entry_value = StringVar(root, value="")
        self.phone_entry = ttk.Entry(root,
                                   textvariable=self.phone_entry_value)
        self.phone_entry.grid(row=1, column=7, padx=5, pady=10, sticky=W)

        birth_label = Label(root, text='Birth')
        birth_label.grid(row=1, column=8, padx=5, pady=10, sticky=W)
        self.birth_entry_value = StringVar(root, value="")
        self.birth_entry = ttk.Entry(root,
                                   textvariable=self.birth_entry_value)
        self.birth_entry.grid(row=1, column=9, padx=5, pady=10, sticky=W)

        # ----- 3RD ROW -----
        sex_label = Label(root, text='Sex')
        sex_label.grid(row=2, column=0, padx=5, pady=10, sticky=W)
        self.sex_entry_value = StringVar(root, value="")
        self.sex_entry = ttk.Entry(root,
                                   textvariable=self.sex_entry_value)
        self.sex_entry.grid(row=2, column=1, padx=5, pady=10, sticky=W)

        lunch_label = Label(root, text='Lunch')
        lunch_label.grid(row=2, column=2, padx=5, pady=10, sticky=W)
        self.lunch_entry_value = StringVar(root, value="")
        self.lunch_entry = ttk.Entry(root,
                                   textvariable=self.lunch_entry_value)
        self.lunch_entry.grid(row=2, column=3, padx=5, pady=10, sticky=W)

        # Create the button that will be used in the next video to add
        # student data to the database
        add_button = ttk.Button(root, text='Add Student', command=self.add_student)
        add_button.grid(column=4, row=2, sticky=(W, E))

        update_button = ttk.Button(root, text='Update Student', command=self.update_student)
        update_button.grid(column=5, row=2, sticky=(W, E))

        delete_button = ttk.Button(root, text='Delete Student', command=self.delete_student)
        delete_button.grid(column=6, row=2, sticky=(W, E))

        # ----- TREEVIEW -----
        # Treeviews can be used to display tables of data
        # Define the column names
        self.tree = ttk.Treeview(root, height=15, columns=('ID', 'First Name', 'Last Name', 'Email',
'Street', 'City', 'State', 'Zip', 'Phone', 'Birth', 'Sex', 'Lunch'), selectmode='browse')

        # Place the tree in the remaining space in the grid
        self.tree.grid(row=3, column=0, columnspan=17)
        # Define that we want to show the heading row
        self.tree['show'] = 'headings'
```

```
        # Assign the heading and column options
        i = 1
        for col in self.headers:
            num = f'#{i}' # Format string to produce incrementing numbers
            self.tree.heading(num, text=col)
            self.tree.column(num, width=115)
            i += 1

        # Create new treeview items and place them in the treeview
        # We get the values to add by cycling through the student
        # data list
        for stud_info in self.student_info:
            num = f'#{i}'
            self.tree.insert('', 'end', values=stud_info)
            i += 1

    def add_student(self):
        pass

    def update_student(self):
        pass

    def delete_student(self):
        pass

# Create the main window
root = Tk()
# Define the size of the main window
root.geometry("1400x600")
# Add a title to our app
root.title("Student Database")
# Create the studentDB object
student_db = StudentDB()
# Continue running our app until quit is clicked
root.mainloop()
```

---------- STUDENT_DB2.PY ----------

```
from tkinter import *
from tkinter import ttk
# NEW
import mysql.connector
from mysql.connector import Error
from datetime import datetime

class StudentDB:
    headers = ['ID', 'First Name', 'Last Name', 'Email', 'Street', 'City', 'State', 'Zip', 'Phone',
'Birth', 'Sex', 'Lunch']
    student_info = []

    # DB connection
    conn = 0
    # Cursor used to traverse results
```

```
    cursor = 0
    # Stores results of last query
    query = 0

    def __init__(self):
        self.tree = None
        self.setup_db() # NEW
        self.create_widgets()

    # NEW
    def setup_db(self):
        try:
            self.conn = mysql.connector.connect(host='localhost', database='students', user='stu-
dentadmin', password='TurtleDove')
        except mysql.connector.Error as error:
            print("Error :", error)

    def create_widgets(self):
        # ----- ROW 1 -----
        sid_label = Label(root, text='ID')
        sid_label.grid(row=0, column=0, padx=5, pady=10, sticky=W)
        self.sid_entry_value = StringVar(root, value="")
        self.sid_entry = ttk.Entry(root,
                                   textvariable=self.sid_entry_value)
        self.sid_entry.grid(row=0, column=1, padx=5, pady=10, sticky=W)

        f_name_label = Label(root, text='First Name')
        f_name_label.grid(row=0, column=2, padx=5, pady=10, sticky=W)
        self.f_name_entry_value = StringVar(root, value="")
        self.f_name_entry = ttk.Entry(root,
                                      textvariable=self.f_name_entry_value)
        self.f_name_entry.grid(row=0, column=3, padx=5, pady=10, sticky=W)

        l_name_label = Label(root, text='Last Name')
        l_name_label.grid(row=0, column=4, padx=5, pady=10, sticky=W)
        self.l_name_entry_value = StringVar(root, value="")
        self.l_name_entry = ttk.Entry(root,
                                      textvariable=self.l_name_entry_value)
        self.l_name_entry.grid(row=0, column=5, padx=5, pady=10, sticky=W)

        email_label = Label(root, text='Email')
        email_label.grid(row=0, column=6, padx=5, pady=10, sticky=W)
        self.email_entry_value = StringVar(root, value="")
        self.email_entry = ttk.Entry(root,
                                     textvariable=self.email_entry_value)
        self.email_entry.grid(row=0, column=7, padx=5, pady=10, sticky=W)

        street_label = Label(root, text='Street')
        street_label.grid(row=0, column=8, padx=5, pady=10, sticky=W)
        self.street_entry_value = StringVar(root, value="")
        self.street_entry = ttk.Entry(root,
                                      textvariable=self.street_entry_value)
        self.street_entry.grid(row=0, column=9, padx=5, pady=10, sticky=W)
```

```
# ----- 2nd ROW -----
city_label = Label(root, text='City')
city_label.grid(row=1, column=0, padx=5, pady=10, sticky=W)
self.city_entry_value = StringVar(root, value="")
self.city_entry = ttk.Entry(root,
                    textvariable=self.city_entry_value)
self.city_entry.grid(row=1, column=1, padx=5, pady=10, sticky=W)

state_label = Label(root, text='State')
state_label.grid(row=1, column=2, padx=5, pady=10, sticky=W)
self.state_entry_value = StringVar(root, value="")
self.state_entry = ttk.Entry(root,
                    textvariable=self.state_entry_value)
self.state_entry.grid(row=1, column=3, padx=5, pady=10, sticky=W)

zip_label = Label(root, text='Zip Code')
zip_label.grid(row=1, column=4, padx=5, pady=10, sticky=W)
self.zip_entry_value = StringVar(root, value="")
self.zip_entry = ttk.Entry(root,
                    textvariable=self.zip_entry_value)
self.zip_entry.grid(row=1, column=5, padx=5, pady=10, sticky=W)

phone_label = Label(root, text='Phone')
phone_label.grid(row=1, column=6, padx=5, pady=10, sticky=W)
self.phone_entry_value = StringVar(root, value="")
self.phone_entry = ttk.Entry(root,
                    textvariable=self.phone_entry_value)
self.phone_entry.grid(row=1, column=7, padx=5, pady=10, sticky=W)

birth_label = Label(root, text='Birth')
birth_label.grid(row=1, column=8, padx=5, pady=10, sticky=W)
self.birth_entry_value = StringVar(root, value="")
self.birth_entry = ttk.Entry(root,
                    textvariable=self.birth_entry_value)
self.birth_entry.grid(row=1, column=9, padx=5, pady=10, sticky=W)

# ----- 3RD ROW -----
sex_label = Label(root, text='Sex')
sex_label.grid(row=2, column=0, padx=5, pady=10, sticky=W)
self.sex_entry_value = StringVar(root, value="")
self.sex_entry = ttk.Entry(root,
                    textvariable=self.sex_entry_value)
self.sex_entry.grid(row=2, column=1, padx=5, pady=10, sticky=W)

lunch_label = Label(root, text='Lunch')
lunch_label.grid(row=2, column=2, padx=5, pady=10, sticky=W)
self.lunch_entry_value = StringVar(root, value="")
self.lunch_entry = ttk.Entry(root,
                    textvariable=self.lunch_entry_value)
self.lunch_entry.grid(row=2, column=3, padx=5, pady=10, sticky=W)

add_button = ttk.Button(root, text='Add Student', command=self.add_student)
add_button.grid(column=4, row=2, sticky=(W, E))
```

```
        update_button = ttk.Button(root, text='Update Student', command=self.update_student)
        update_button.grid(column=5, row=2, sticky=(W, E))

        delete_button = ttk.Button(root, text='Delete Student', command=self.delete_student)
        delete_button.grid(column=6, row=2, sticky=(W, E))

        # ----- TREEVIEW -----
        self.tree = ttk.Treeview(root, height=15, columns=('ID', 'First Name', 'Last Name', 'Email',
'Street', 'City', 'State', 'Zip', 'Phone', 'Birth', 'Sex', 'Lunch'), selectmode='browse')

        self.tree.grid(row=3, column=0, columnspan=17)
        self.tree['show'] = 'headings'

        i = 1
        for col in self.headers:
            num = f'#{i}' # Format string to produce incrementing numbers
            self.tree.heading(num, text=col)
            self.tree.column(num, width=115)
            i += 1

        self.update_table()

    # Check that there is an entry in all entries required
    # Verify if student id is required
    def all_entries_filled(self, sid_required):
        if len(self.f_name_entry_value.get()) == 0 or len(self.l_name_entry_value.get()) == 0 or
len(self.email_entry_value.get()) == 0 or len(self.street_entry_value.get()) == 0 or len(self.city_en-
try_value.get()) == 0 or len(self.state_entry_value.get()) == 0 or len(self.zip_entry_value.get()) ==
0 or len(self.phone_entry_value.get()) == 0 or len(self.birth_entry_value.get()) == 0 or len(self.-
sex_entry_value.get()) == 0 or len(self.lunch_entry_value.get()) == 0:
            return False
        elif sid_required:
            if len(self.sid_entry_value.get()) == 0:
                return False
            else:
                return True
        else:
            return True

    # NEW Executes the query and fetches result from
    # the query if it is expected
    def execute_query(self, result_expected):
        try:
            # Get connection for cursor
            self.cursor = self.conn.cursor()
            self.cursor.execute(self.query)
            # Check if a result is expected from the query
            if result_expected:
                self.student_info = self.cursor.fetchall()

            # Move changes to DB
            self.conn.commit()
            # Reset results and close the cursor
            self.cursor.close()
```

```
        except mysql.connector.Error as error:
            print("Error :", error)

    # NEW clear the tree and then get updated data
    def update_table(self):
        for i in self.tree.get_children():
            self.tree.delete(i)

        # Get all student data for the treeview
        self.query = "SELECT student_id, first_name, last_name, email, street, city, state, zip,
phone, birth_date, sex, lunch_cost FROM students"

        self.execute_query(True)

        i = 1
        for stud_info in self.student_info:
            num = f'#{i}'
            self.tree.insert('', 'end', values=stud_info)
            i += 1

    def add_student(self):
        # Check if connected to DB and all required entries are filled
        if(self.conn.is_connected() and not self.all_entries_filled(False)):
            self.popup_msg("Enter All the Student Data")
        else:
            # Get the current time and format it to fit what
            # MySQL expects
            now_time = datetime.now()
            format_date = now_time.strftime('%Y-%m-%d %H:%M:%S')
            f_name = self.f_name_entry_value.get()
            l_name = self.l_name_entry_value.get()
            email = self.email_entry_value.get()
            street = self.street_entry_value.get()
            city = self.city_entry_value.get()
            state = self.state_entry_value.get()
            zip = self.zip_entry_value.get()
            phone = self.phone_entry_value.get()
            birth = self.birth_entry_value.get()
            sex = self.sex_entry_value.get()
            lunch = self.lunch_entry_value.get()

            self.query = f"INSERT INTO students VALUES( NULL, '{f_name}', '{l_name}', '{email}',
'{street}', '{city}', '{state}', {zip}, '{phone}', '{birth}', '{sex}', '{format_date}', {lunch})"

            self.execute_query(False)

            # Update the table
            self.update_table()

    def update_student(self):
        if(self.conn.is_connected() and not self.all_entries_filled(True)):
            self.popup_msg("Enter All the Student Data")
        else:
            sid = self.sid_entry_value.get()
```

```
            f_name = self.f_name_entry_value.get()
            l_name = self.l_name_entry_value.get()
            email = self.email_entry_value.get()
            street = self.street_entry_value.get()
            city = self.city_entry_value.get()
            state = self.state_entry_value.get()
            zip = self.zip_entry_value.get()
            phone = self.phone_entry_value.get()
            birth = self.birth_entry_value.get()
            sex = self.sex_entry_value.get()
            lunch = self.lunch_entry_value.get()

            self.query = f"UPDATE students SET first_name = '{f_name}', last_name = '{l_name}',
email = '{email}', street = '{street}', city = '{city}', state = '{state}', zip = {zip}, phone = '{phone}',
birth_date = '{birth}', sex = '{sex}', lunch_cost = {lunch} WHERE student_id = {sid}"
            self.execute_query(False)

            # Update the table and don't ask for a result
            self.update_table()

    def delete_student(self):
        if(self.conn.is_connected()):
            if len(self.sid_entry_value.get()) == 0:
                self.popup_msg("Enter A Student ID Number")
            else:
                sid = self.sid_entry_value.get()
                self.query = f"DELETE FROM students WHERE student_id = {sid}"
                self.execute_query(False)

                # Update the table
                self.update_table()

    # Used to create a popup dialog
    def popup_msg(self, msg):
        popup = Tk()
        popup.geometry("235x85")
        popup.resizable(width=False, height=False)
        popup.wm_title("Enter All Values")
        err_msg = Text(popup, font=("Verdana", 16))
        err_msg.insert(INSERT, msg)
        err_msg.pack()
        ok_but = ttk.Button(popup, text="OK", command=popup.destroy)
        ok_but.place(relx=.5, rely=.8, anchor="center")
        popup.mainloop()

root = Tk()
root.geometry("1400x600")
root.title("Student Database")
student_db = StudentDB()
root.mainloop()
```

Data Structures & Algorithms

Computer Science & the Programmer

1. Computer Science is the science of solving problems. The solution to that problem is called an algorithm. That algorithm is a step-by-step list of instructions that lead to a solution.
2. You can drive a car by understanding the break / gas pedals, steering wheel, turn signal, etc. The manufacturer of that car understands how every part works within the car and provides an easy to understand interface for the user.
3. As a programmer you understand the details and also must provide an easy to understand interface.

Why Study Data Structures

1. We will often want to define custom types of data. We will also need to provide an easy way for users to access that data.
2. By studying the most useful data structures we will get better at designing custom versions

Why Study Algorithms

1. By studying the most effective ways of solving problems we will begin to recognize how to utilize those solutions to solve numerous other problems.

Why is One Algorithm Better

1. Which is more readable
2. Which consumes the least amount of memory
3. Execution time

```
import timeit

def get_sum(max_num):
    sol = 0
    for i in range(1, max_num + 1):
        sol += i
    return sol

def get_sum_2(max_num):
    sol = 0
    i = 1
    while i < max_num:
        sol += i
        i += 1
    return sol

def get_sum_3(mn):
    return mn * (mn + 1) / 2

# Use timeit to verify execution time
# Pass the function, how many times to repeat the timer, number of times to
# execute the function and provide access to the functions
print("Testing get_sum")
print(timeit.repeat(stmt='get_sum(100000)', repeat=5, number=1, globals=globals()))
```

```
# The while loop under performs because more values
# are required to execute it
print("Testing get_sum_2")
print(timeit.repeat(stmt='get_sum_2(100000)', repeat=5, number=1, globals=globals()))

# This is the most efficient algorithm because it doesn't
# contain repeated steps which allows it to perform well
# as the number of values summed dramatically increases
print("Testing get_sum_3")
print(timeit.repeat(stmt='get_sum_3(100000)', repeat=5, number=1, globals=globals()))

# Simply calculating time is not the best way to judge
# an algorithms performance because it is largely
# based on the computer / language using that algorithm

# It is better to quantify performance based on the number
# of steps the algorithm requires.

# Big-O Notation, where the O refers to the Order of
# Magnitude, is a measure of how well an algorithm
# scales as the amount of data increases

# How well does it perform as it increases from a 10
# element array versus a 10,000 element array

# Let's say you have this algorithm
# 45n^3 + 20n^2 + 19 = 84 (if n is 1)
# I want to define the part of the algorithm that has
# the biggest effect during the calculation of the answer
# If n=2 the answer goes from 84 to 459
# It doesn't take long until + 19 doesn't matter
# If n=10 the answer is 47,019 and n^2 has little effect
# on our answer because 45n^3 = 45,000
# Because the n^3 has the greatest effect on our final
# answer and so we would say this algorithm has an order
# of n^3 or O(N^3)

# I'll cover what all of these mean
# O(1), O(N), O(N^2), O(log N), O(N log N)
```

Data Structures & Algorithms 2

```
import random
import time

list_1 = []
start_time = 0
end_time = 0

def generate_rand_list(max_size):
    new_list = []
    for i in range(0, max_size):
```

```
        new_list.append(random.randint(1, 100))
    return new_list

list_1 = generate_rand_list(10)

# O(1) : An algorithm that executes in the same amount
# of time regardless of how big the list is
# A 10 item list or a 10,000 item list will always
# take the same amount of time with this operation
def add_item_to_list(num):
    list_1.append(num)

# O(N) : Algorithm thats time to complete is directly
# proportional to the amount of data supplied
# An example of this is a linear search because it
# requires us to look in each space of an array
# This is true even if we find the item during the 1st
# search because Big-O Notation describes the worst
# case situation through using the algorithm
def linear_search(val):
    val_found = "Value Not Found"
    for i in list_1:
        if i == val:
            val_found = "Value Found"
    print(val_found)


# print("Testing Linear Search")
# list_1 = generate_rand_list(10)
# start_time = time.time()
# linear_search(10000)
# print(f"{time.time() - start_time} seconds")
#
# list_1 = generate_rand_list(1000)
# start_time = time.time()
# linear_search(10000)
# print(f"{time.time() - start_time} seconds")
#
# list_1 = generate_rand_list(10000)
# start_time = time.time()
# linear_search(10000)
# print(f"{time.time() - start_time} seconds")
#
# list_1 = generate_rand_list(100000)
# start_time = time.time()
# linear_search(10000)
# print(f"{time.time() - start_time} seconds")

# O(N^2) : Algorithms thats time to complete is
# proportional to the square of the amount of data.
# A Bubble sort is an example because it contains
# nested iterations. Further nested iterations
# will result in O(N^3), O(N^4) performance
# Each pass through the outer loop O(N) requires
```

```
# us to go through the entire list again so N is
# squared

# The Bubble sort is a way to sort a list
# It works this way
# 1. An outer loop decreases in size each time
# 2. The goal is to have the largest number at the end of the list when the outer loop completes
1 cycle
# 3. The inner loop starts comparing indexes at the beginning of the loop
# 4. Check if list[Index] > list[Index + 1]
# 5. If so swap the index values
# 6. When the inner loop completes the largest number is at the end of the list
# 7. Decrement the outer loop by 1


def bubble_sort():
    list_size = len(list_1)
    # Cycle through each value in the list
    for i in range(list_size):
        # We don't have to check previous i items
        # checked
        for j in range(0, list_size-i-1):
            # If the 1st value is greater then the
            # 2nd have them swap
            if list_1[j] > list_1[j+1]:
                list_1[j], list_1[j+1] = list_1[j+1], list_1[j]

            # Demonstrates how the Bubble sort works
            # for k in list_1:
            #     print(k, end=", ")
            # print()


# Shows Bubble Sort changes as it cycles
# list_1 = generate_rand_list(10)
# bubble_sort()

# Bubble Sort Tests
# list_1 = generate_rand_list(100)
# start_time = time.time()
# bubble_sort()
# print(f"{time.time() - start_time} seconds")
#
# list_1 = generate_rand_list(1000)
# start_time = time.time()
# bubble_sort()
# print(f"{time.time() - start_time} seconds")
#
# list_1 = generate_rand_list(10000)
# start_time = time.time()
# bubble_sort()
# print(f"{time.time() - start_time} seconds")
```

Data Structures & Algorithms 3

```
# O(log N) : Algorithms in which the amount of data
# is roughly decreased by 50% each time through
# the algorithm.

# If 10^2 = 100 then the log10(100) = 2, because
# a logarithm tells us what power is used to make
# a number

# Log N increases at a dramatically slower rate as
# N increases which makes them more efficient as
# N increases. [Slide]

# A Binary search is an example.
# The Binary Search is extremely fast, but the
# negative is that it only works with sorted lists.
# After the sort it starts searching in the middle of
# the list which allows it to eliminate 1/2 of the
# values after each cycle through the list.

# def binary_search(value: int):
#     list_size = len(list_1)
#     low_index = 0
#     high_index = list_size - 1
#     while low_index <= high_index:
#         mid_index = int((high_index + low_index) / 2)
#         if list_1[mid_index] < value:
#             low_index = mid_index + 1
#         elif list_1[mid_index] > value:
#             high_index = mid_index - 1
#         else:
#             print(f"Found a match for {value} at index {mid_index}")
#             low_index = high_index + 1
#
#
# list_1 = generate_rand_list(1000)
# bubble_sort()
# start_time = time.time()
# binary_search(150)
# print(f"{time.time() - start_time} seconds")
#
# list_1 = generate_rand_list(10000)
# bubble_sort()
# start_time = time.time()
# binary_search(150)
# print(f"{time.time() - start_time} seconds")

# O(n log n) : The Quick Sort is an example of this
# I'll explain the Quick Sort and then show how
# it matches with this Big-O

# With the Quick Sort we divide up all values in the
```

```
# list into 2 parts. Each part is called a
# partition. The value that lies in the middle of those
# 2 parts is called the pivot. As we cycle through
# the list if a value is greater then the pivot it
# goes to the right and if less it goes to the left.
# Slide


def partition(start, end):
    pivot = list_1[start]
    low = start + 1
    high = end
    while True:
        while low <= high and list_1[high] >= pivot:
            high = high - 1
        while low <= high and list_1[low] <= pivot:
            low = low + 1

        if low <= high:
            list_1[low], list_1[high] = list_1[high], list_1[low]
        else:
            break
    list_1[start], list_1[high] = list_1[high], list_1[start]
    return high


def quick_sort(start, end):
    # Demonstrates how the Quick sort works
    for k in list_1:
        print(k, end=", ")
    print()

    if start >= end:
        return
    part = partition(start, end)
    quick_sort(start, part - 1)
    quick_sort(part + 1, end)


list_1 = [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
quick_sort(0, len(list_1) - 1)
print(list_1)

# Most sorts are O(N) because every element must be
# looked at once. The Bubble Sort as we saw is O(N^2)
# To prove that the Quick Sort is O(n log n) we know
# that values are only compared once. So, each comparison
# will reduce the possible final sorted lists in half.
# So the number of comparisons is log n! (Factorial of N)
# Comparisons = log n + log(n-1) + ... + log(1)
# Evaluates to n log n
```

Data Structures & Algorithms 4

```
import time

# In this tutorial I'll cover the 2 Linear Data Structures
# Stacks and Queues. Next time I'll cover Deques and Lists.
# They differ based on how data is added or removed.

# STACKS : A collection where items are both added and removed
# from the top. This is known as a Last In First Out collection.
# Think of this as a stack of books where the last one added to
# the stack is the 1st to come off.

# Here we implement the functions of a Stack using a List


class Stack:
    def __init__(self):
        self.stack = []

    def push(self, data):
        self.stack.insert(0, data)

    def is_empty(self):
        return self.stack == []

    def pop(self):
        if self.is_empty():
            return "Stack Empty"
        return self.stack.pop(0)

    def peek(self):
        return self.stack[0]

    def size(self):
        return len(self.stack)

    # Lets create a function using our stack that reverses
    # a string
    def reverse_string(self):
        while True:
            if len(self.stack) == 0:
                break
            else:
                print(self.stack.pop(0), end="")


s_1 = Stack()
s_1.push("Cat")
s_1.push("Dog")
print(s_1.pop())
print(s_1.pop())
print(s_1.pop())
```

```
# Reverses the characters passed in
s_1.push("C")
s_1.push("a")
s_1.push("t")
s_1.reverse_string()


# QUEUES : A collection that operates using First In
# First Out logic. An example of this in the real
# world would be a line. If you are the 1st there
# you are served 1st.

# Let's make a Queue using a List

class Queue:
    def __init__(self):
        self.queue = []

    def enqueue(self, data):
        self.queue.insert(0, data)

    def is_empty(self):
        return self.queue == []

    def dequeue(self):
        if self.is_empty():
            return "Stack Empty"
        return self.queue.pop()

    def size(self):
        return len(self.queue)

    # This function will pause as each item has
    # its turn
    def wait_your_turn(self):
        while True:
            if len(self.queue) == 0:
                break
            else:
                print(f"{self.dequeue()} takes their turn")
                # Pause 3 seconds
                time.sleep(3)


print()
q_1 = Queue()
q_1.enqueue("Cat")
q_1.enqueue("Dog")
print(q_1.dequeue())
print(q_1.dequeue())
print(q_1.dequeue())

# Test turn taking function
```

```
q_1.enqueue("Cat")
q_1.enqueue("Dog")
q_1.wait_your_turn()

# Next time I'll cover Deques and Lists
```

Data Structures & Algorithms 5

```
# In this tutorial I'll cover the other 2 Linear Data Structures
# being Deques and Linked Lists.

# DEQUES : A collection that allows you to add or remove data
# from either the front or end of the list.


class Deque:
    def __init__(self):
        self.deque = []

    def add_front(self, data):
        self.deque.append(data)

    def add_rear(self, data):
        self.deque.insert(0, data)

    def is_empty(self):
        return self.deque == []

    def remove_front(self):
        if self.is_empty():
            return "Empty Deque"
        else:
            return self.deque.pop()

    def remove_rear(self):
        if self.is_empty():
            return "Empty Deque"
        else:
            return self.deque.pop(0)

    def size(self):
        return len(self.deque)

    # Checks if string is a palindrome which
    # is the same word forward or backward
    # Racecar, Rotator, etc.
    def check_palindrome(self):
        is_palindrome = True
        while self.size() > 1 and is_palindrome:
            front = self.remove_front()
            rear = self.remove_rear()
            if front != rear:
                is_palindrome = False
```

```
        return is_palindrome



d_1 = Deque()
d_1.add_front("Dog")
d_1.add_rear("Cat")
d_1.add_rear("Mouse")
print(f"Front : {d_1.remove_front()}")
print(f"Rear : {d_1.remove_rear()}")
print(f"Size : {d_1.size()}")

# Check for palindrome
d_2 = Deque()
word = "racecar"
for i in word:
    d_2.add_rear(i)
print(f"Palindrome : {d_2.check_palindrome()}")

word_2 = "zero"
for i in word_2:
    d_2.add_rear(i)
print(f"Palindrome : {d_2.check_palindrome()}")

# LINKED LIST : A collection in which each item is only
# aware of the next item in the list. The last item in
# the list is also aware that there is no more values.
# Linked lists refer to each item in the list as a Node.

class Node:
    def __init__(self, data):
        self.data = data
        # Each node starts with no reference to the next
        self.next = None

    # Retrieves data stored
    def get_data(self):
        return self.data

    # Changes data stored
    def set_data(self, new_data):
        self.data = new_data

    # Stores the next node in the list
    def set_next(self, new_next):
        self.next = new_next

    # Retrieves the next node in the list
    def get_next(self):
        return self.next


class LinkedList:
    # The LinkedList will be assigned the 1st Node
```

```
def __init__(self):
    self.head = None

# We'll have to check if a head node exists
def is_empty(self):
    return self.head is None

# Adds nodes to the list by
# 1. Creating a new node and assigning data
# 2. Setting next as the previous head node
# 3. Making the new node the lists new head node
def add(self, data):
    new_node = Node(data)
    new_node.set_next(self.head)
    self.head = new_node

# Removing a node requires us to :
# 1. Start at the head node
# 2. Check if it has the data we are searching for
# 3. If not search for the next node in the list
# 4. Repeat checking for data as long as nodes are left
def remove(self, search_value):
    current_node = self.head
    # As we cycle this stores the previous node searched
    # so we can use it to find the next node in the list
    prev_node = None
    data_found = False

    # Check if matching data exists
    while not data_found:
        if current_node.get_data() == search_value:
            data_found = True
        # If not use the previously checked node to find
        # the next node in the list
        else:
            prev_node = current_node
            current_node = prev_node.get_next()
    # Assign current nodes next node to head
    if prev_node is None:
        self.head = current_node.get_next()
    else:
        # Assign the next node
        prev_node.set_next(current_node.get_next())

# We could increment a length value each time a
# new node is added, or we could cycle through
# the LinkedList until get_next returns None
def length(self):
    # Start cycling at the head
    current_node = self.head
    # Stores number of nodes
    total_nodes = 0
    # Cycle until the next node in the list = None
    while current_node is not None:
```

```
            total_nodes += 1
            current_node = current_node.get_next()
        return total_nodes

    # Searches for a value in the LinkedList and returns
    # True or False
    def search(self, search_value):
        current_node = self.head
        data_found = False
        # Cycle through LinkedList, skipping to next node
        # along the way until you find a match
        while current_node is not None and not data_found:
            if current_node.get_data() == search_value:
                data_found = True
            else:
                current_node = current_node.get_next()
        return data_found


ll = LinkedList()
ll.add(1)
ll.add(2)
print(f"Length : {ll.length()}")
print(f"1 : {ll.search(1)}")
ll.remove(1)
print(f"Length : {ll.length()}")
print(f"1 : {ll.search(1)}")
```

Data Structures & Algorithms 6

```
# Previously I covered both Linear and Binary Search.
# This time I will cover hashing. Through hashing our
# goal is to store data in such a way as we'll be able
# to search in O(1) time.

# The positive of a Hash Table data structure is that
# they are fast at inserting and searching. The negative
# is that they are limited in size, are hard to resize,
# and don't work well unless each item is unique.

# A Hash Function is used to generate a unique key for
# every item in the list. Since every item is entered
# using a calculation, we can reverse the calculation
# to find the correct index.

# We have to take care that the unique key (Index) fits
# in the list and that it doesn't overwrite other data.


class HashFunction:

    def __init__(self, size):
```

```
    self.list_size = size
    self.the_list = []
    for i in range(size):
        self.the_list.append("-1")

# This function gets a list of strings that are
# numbers and stores them in a matching value index
# This won't create a good Hash Table, but it is
# a Hash Table
def hash_func_1(self, str_list):
    for j in str_list:
        # Cast string value to integer
        index = int(j)
        # Store value where index and value match
        self.the_list[index] = j

# For our next Hash Function we have to have values
# from 0 to 999, but we will have a max of 15 values
# It doesn't make sense to make a 1000 item list
# We can however use the mod function versus the
# list size to make sure the items fit in our list.
# Also our goal is to make the list big enough to
# avoid collisions but not so big that it wastes
# memory. A collision occurs when we try to put a
# value in an index that already has data stored.
def hash_func_2(self, str_list_2):
    for k in str_list_2:
        str_int = int(k)
        index = str_int % 29
        print(f"Mod Index : {index} Value : {str_int}")

        # Look for a collision
        while self.the_list[index] != "-1":
            index += 1
            print(f"Collision Try {index} Instead")
            # If we get to the end of the list go to index 0
            index %= self.list_size

        # We know we found an index where we can store
        self.the_list[index] = k

# This function will find a value in a Hash table and return
# the index for it
def find_key(self, key):
    # Use the same formula used to store the value
    list_index_hash = int(key) % 29

    # Cycle through our list looking for the value and
    # then return the index. If the value was moved
    # because there wasn't enough room continue searching
    # using the same formula as before
    while self.the_list[list_index_hash] != "-1":
        if self.the_list[list_index_hash] == key:
            print(f"{key} in Index {list_index_hash}")
```

```
            return self.the_list[list_index_hash]

        # If not found look in next index
        list_index_hash += 1

        # If we get to the end of the list go to index 0
        list_index_hash %= self.list_size

    # If we are here that means we couldn't find it
    return False


# This is a bad Hash Function built around storing based
# on the value of the string to be stored
hash_table = HashFunction(30)
# This list can't have a value greater then 29
str_list = ["1", "5", "17", "21", "26"]
# Pass in the list to sort it in the Hash Table
hash_table.hash_func_1(str_list)
# Print the new list with indexes
for i in range(hash_table.list_size):
    print(i, end=" ")
    print(hash_table.the_list[i])

# This hash function stores items by creating a constrained
# index, but also based on the value of the string
# This was constructed to cause collisions, but normally
# you want your list to be twice the size the max number of
# values you plan to add
hash_table_2 = HashFunction(30)
str_list_2 = ["100", "510", "170", "214", "268", "398",
              "235", "802", "900", "723", "699", "1", "16",
              "999", "890", "725", "998", "990", "989", "984",
              "320", "321", "400", "415", "450", "50", "660", "624"]
hash_table_2.hash_func_2(str_list_2)
# Print the new list with indexes
for i in range(hash_table_2.list_size):
    print(i, end=" ")
    print(hash_table_2.the_list[i])

# We will search for what index our data is stored in
hash_table_2.find_key("660")
```

Data Structures & Algorithms 7

```
# What I cover in this tutorial :
# Why we should use Primes when constructing Hash Tables
# How to increase hash table size even though I said to avoid it
# What clustering is and how to avoid it
# How to work with Double Hashing

# Why We Use Primes?
# Previously we calculated the index by using the value to
```

```
# store and shrunk it down to fit in the list using modulus
# We want to avoid collisions which are caused when we
# try to store similar data.
# N values with similar data actually cause N times the
# number of collisions.
# If we use lists with the size of a prime we can cut down
# on collisions.

class HashFunctions:
    def __init__(self, size):
        self.list_size = size
        self.the_list = []
        for i in range(size):
            self.the_list.append(None)

    # This is the hash function from the previous tut
    # Let's compare the number of collisions versus
    # using 30 and then 31 which is a prime number
    def hash_func_2(self, str_list):
        for k in str_list:
            str_int = int(k)
            index = str_int % 31
            # print(f"Mod Index : {index} Value : {str_int}")

            # Look for a collision
            while self.the_list[index] is not None:
                index += 1
                # print(f"Collision Try {index} Instead")
                # If we get to the end of the list go to index 0
                index %= self.list_size

            # We know we found an index where we can store
            self.the_list[index] = k

    # All prime numbers except 2 & 3 are of the form 6k +/- 1
    def is_prime(self, num):
        # 0 & 1 are Not Prime
        if num <= 1:
            return False
        # 2 & 3 are Prime
        if num <= 3:
            return True
        # We can eliminate values up to 25 by just checking
        # for divisibility of 2 or 3
        if num % 2 == 0 or num % 3 == 0:
            # print(f"{num} Divisible by 2 or 3")
            return False

        # We will test if num is not 6 * num +/- 1 and then increment
        # j by 6 each time
        j = 5

        # Let's square j instead of getting the square root of
        # num to avoid using the math module
```

```
    while j * j <= num:
        # print(f"J : {j} num : {num}")
        # print(f"num % j == 0 : {num % j}")
        # print(f"num % j + 2 == 0 : {num % (j + 2)}")
        # We only need to test if the number is divisible by
        # 5 & 7, 11 & 13, 17 & 19, ... because a prime
        # 6 * num +/- 1
        if num % j == 0 or num % (j + 2) == 0:
            return False

        # Increment to check the next grouping
        j = j + 6
    return True

# Now that I can get primes I need a function that will
# generate the next prime that is greater than the
# minimum list size required
def get_next_prime(self, min_size):
    while True:
        if self.is_prime(min_size):
            return min_size
        else:
            min_size += 1

# Function that finds the next required array size
def increase_list_size(self, min_size):
    new_list_size = self.get_next_prime(min_size)
    self.move_old_list(new_list_size)

# This function will clear all values in the main list
def fill_list_with_none(self):
    for k in range(self.list_size):
        self.the_list.append(None)

# Removes moves all values in list to the beginning
# of the list
def remove_empty_spaces_in_list(self):
    temp_list = []

    # if a list item isn't None add it to the temp_list
    for j in self.the_list:
        if j is not None:
            temp_list.append(j)

    return temp_list

def move_old_list(self, new_list_size):
    # Update list size
    self.list_size = new_list_size

    # Store old list in a new one with spaces removed
    clean_list = self.remove_empty_spaces_in_list()

    # Fill list with None values
```

```
        self.fill_list_with_none()

        # Store the same 30 items again in our new list
        self.hash_func_2(clean_list)


l_2 = ["30", "60", "90", "120", "150", "180", "210", "240", "270", "300", "330", "360", "390",
"420", "450", "480",
       "510", "540", "570", "600", "989", "984", "320", "321", "400", "415", "450", "50", "660",
"624"]

hash_func = HashFunctions(31)
hash_func.hash_func_2(l_2)
for i in range(hash_func.list_size):
    print(i, end=" ")
    print(hash_func.the_list[i])

# print("Find Primes")
# for i in range(500):
#     if hash_func.is_prime(i):
#         print(i)

hash_func.increase_list_size(60)
for i in range(hash_func.list_size):
    print(i, end=" ")
    print(hash_func.the_list[i])
```

Data Structures & Algorithms 8

```
# In this video we will focus on avoiding clustering
# Since each time we have a collision we just move
# down 1 index that causes large clusters of data.
# The more collisions we have increases the odds of
# more collisions. That is why we'll have both large
# empty parts as well as large clusters in our lists.

class HashFunctions:
    def __init__(self, size):
        self.list_size = size
        self.the_list = []
        for i in range(size):
            self.the_list.append("-1")

    # Let'd change our hash function to avoid
    # cluster creation
    # We will do this by creating a double hash
    def dbl_hash_func(self, str_list):
        for k in str_list:
            str_int = int(k)
            index = str_int % 61
            # Look for a collision and if found go to the next available
            # This is bad because it causes clustering.
```

```
        # We can avoid that by changing the step distance
        # using a prime number.
        # That step will be between 1 and 7
        # If this list was very large you will see much more clustering
        step_distance = 7 - (int(self.the_list[index]) % 7)

        while self.the_list[index] != "-1":

            # Replace this
            # index += 1
            # with this
            index += step_distance

            # At end of list -> go to index 0
            index %= self.list_size
        # We have a place to store
        self.the_list[index] = k
        self.print_list()
        print()

    def print_list(self):
        # Used to print each row of columns 10 at a time
        increment = 0
        # I want to allow for 10 columns per row
        num_of_rows = int((self.list_size / 10) + 1)
        for j in range(num_of_rows):
            self.print_line(78)
            # Get the next row of columns to print
            increment += 10
            # Print a row of indexes and then a row of data
            self.print_row(increment, False)
            self.print_line(78)
            self.print_row(increment, True)
        self.print_line(78)

    # Print a horizontal line for the table
    def print_line(self, num_of_lines):
        for l in range(num_of_lines):
            print("-", end="")
        print()

    # Used to print indexes and data for the table
    # Receives the row of data to print and whether it is data
    # or indexes
    def print_row(self, increment, is_data):
        k = increment - 10
        while k <= increment:
            # If past the end of the array print blank spaces
            if k > self.list_size - 1:
                print("|     ", end=" ")
            else:
                if not is_data:
                    # Print value with 3 spaces and right justify
                    # Print index numbers
```

```
                print("| {:>3} ".format(k), end=" ")
            else:
                # Print list data values or nothing if -1
                if self.the_list[k] == "-1":
                    print("| {:>3} ".format(" "), end=" ")
                else:
                    print("| {:>3} ".format(self.the_list[k]), end=" ")
        k += 1
    print("|")

# Now we need to update our find function using double hashing
def find_key(self, key):
    # Use the same formula used to store the value
    list_index_hash = int(key) % 61

    # NEW Add this to calculate step distance
    step_distance = 7 - (int(self.the_list[list_index_hash]) % 7)

    # Cycle through our list looking for the value and
    # then return the index
    while self.the_list[list_index_hash] != "-1":
        if self.the_list[list_index_hash] == key:
            print(f"{key} in Index {list_index_hash}")
            return self.the_list[list_index_hash]

        # NEW We change this
        # list_index_hash += 1
        # to this
        list_index_hash += step_distance

        # If we get to the end of the list go to index 0
        list_index_hash %= self.list_size

    # If we are here that means we couldn't find it
    return False


l_2 = ["100", "510", "170", "214", "268", "398",
       "235", "802", "900", "723", "699", "1", "16",
       "999", "890", "725", "998", "990", "989", "984",
       "320", "321", "400", "415", "450", "50", "660", "624"]

hash_func = HashFunctions(61)
hash_func.dbl_hash_func(l_2)
hash_func.find_key("170")
```

Data Structures & Algorithms 9

```
# In this tutorial I'm going to cover hashing strings

# When hashing strings we convert it into a number.
# We get the character code for each character
# (hash_val * 27 (total number of characters) + char_code)
# % array_size

class HashFunction:
    def __init__(self, size):
        self.list_size = size
        self.the_list = []
        for i in range(size):
            self.the_list.append("-1")

    def hash_string(self, str_to_hash):
        # Holds character hash value
        hash_val = 0
        # Holds sum of character hash values
        hash_sum = 0
        for i in range(len(str_to_hash)):
            # Character code for a is 97 so I'll subtract
            # 96 from it so we can start at 1
            # ord() returns the character code
            char_code = ord(str_to_hash[i]) - 96

            # Temporarily store hash key value
            hkv_temp = hash_val

            # Hash this character like we did before
            hash_sum += (hash_val * 27 + char_code)

        return hash_sum

    def hash_str_list(self, str_list):
        for str_to_hash in str_list:
            # Hash the individual string
            hash_sum = self.hash_string(str_to_hash)
            # Constrain the hash_value to the size of the list
            hash_sum = hash_sum % self.list_size

            # Used to avoid clustering
            step_distance = 7 - (hash_sum % 7)

            # Continue cycling until we find a -1 and then
            # place data there
            while self.the_list[hash_sum] != "-1":
                hash_sum += step_distance
                hash_sum %= self.list_size

            self.the_list[hash_sum] = str_to_hash
```

```
def find(self, value):
    value_index = self.hash_string(value)
    step_distance = 7 - (value_index % 7)

    # Cycle through our list looking for the value and
    # then return the index
    while self.the_list[value_index] != "-1":
        if self.the_list[value_index] == value:
            print(f"{value} in Index {value_index}")
            return self.the_list[value_index]

        value_index += step_distance

        value_index %= self.list_size

        # If we are here that means we couldn't find it
    return False

def print_list(self):
    # Used to print each row of columns 10 at a time
    increment = 0
    # I want to allow for 10 columns per row
    num_of_rows = int((self.list_size / 10) + 1)
    for j in range(num_of_rows):
        self.print_line(78)
        # Get the next row of columns to print
        increment += 10
        # Print a row of indexes and then a row of data
        self.print_row(increment, False)
        self.print_line(78)
        self.print_row(increment, True)
    self.print_line(78)

# Print a horizontal line for the table
def print_line(self, num_of_lines):
    for l in range(num_of_lines):
        print("-", end="")
    print()

# Used to print indexes and data for the table
# Receives the row of data to print and whether it is data
# or indexes
def print_row(self, increment, is_data):
    k = increment - 10
    while k <= increment:
        # If past the end of the array print blank spaces
        if k > self.list_size - 1:
            print("|     ", end=" ")
        else:
            if not is_data:
                # Print value with 3 spaces and right justify
                # Print index numbers
                print("| {:>3} ".format(k), end=" ")
            else:
```

```
                # Print list data values or nothing if -1
                if self.the_list[k] == "-1":
                    print("| {:>3} ".format(" "), end=" ")
                else:
                    print("| {:>3} ".format(self.the_list[k]), end=" ")
            k += 1
        print("|")


words_to_add = ["ace", "act", "add", "age", "ago", "aid", "aim", "air", "all", "amp", "and",
"ant", "any", "ape", "apt", "arc", "are", "ark", "arm", "art", "ash", "ask", "asp", "ate", "atm",
"awe", "axe", "aye"]

hash_func = HashFunction(61)
hash_func.hash_str_list(words_to_add)
hash_func.print_list()
hash_func.find("ask")
```

Data Structures & Algorithms 10

```
# I'll cover Binary Trees, How to create / add / delete nodes,
# tree traversal and how to find nodes.

# Root : Top Node of a Tree
# Path : Lines that Connect Nodes
# Leaf : Node without Children
# Subtree : Grouping of Parent Children in the Main Tree

# What is a Binary Tree
# It is a tree thats nodes will only ever have 2 children and the
# child on the left will have a value < than parent, while
# the child on the right has a value > than the parent.
# Parents can however only have 1 child node

# Unbalanced Trees are trees in which most of the Nodes are
# on one side. They are bad because they are slow.

# Why Use Trees?
# Fast Search, Insert & Delete
# Ordered Arrays are Bad at Insert & Delete
# Linked List Searching is Slow
# Tree Operations are O(log N)
# Trees are More Efficient if Many Different Operations
# are Needed

class BinaryTree:
    def __init__(self):
        # Every tree has a root node
        self.root = None

    def add_node(self, key, name):
        # Create and initialize the Node
        new_node = Node(key, name)
```

```
    # If no Root assign it
    if self.root is None:
        self.root = new_node
    else:
        # Set Root as the Node we will start with as
        # we traverse the tree
        focus_node = self.root

        while True:
            # Root is the top parent so we start there
            parent = focus_node

            # Check if new node should go on the left
            # or right
            if key < focus_node.key:
                # Switch focus to the left child
                focus_node = focus_node.left_child

                # If the left_child has no children
                if focus_node is None:
                    # Place the new node on the left
                    parent.left_child = new_node
                    # All Done with Adding
                    return True
            else:
                # If here put the node on the right
                focus_node = focus_node.right_child
                # If the right child has no children
                if focus_node is None:
                    # Place the new node on the right
                    parent.right_child = new_node
                    return True

# Now that we can add nodes it would be nice to traverse
# our Tree
# In Order Traversal (SLIDE)
# Start with left child, when None jump to Parent and to
# the right child, back to parent and repeat
def in_order_traverse(self, focus_node):
    # We visit nodes in ascending order
    # Recursion is used to go to 1 node and then to its
    # children nodes
    if focus_node is not None:
        self.in_order_traverse(focus_node.left_child)
        print(focus_node)
        self.in_order_traverse(focus_node.right_child)

# With Preorder Traversal we hit our focus node root
# and then cycle through all our left children after
# that we jump up 1 parent and check for a right child
# Then we go back up to root and work our way down
# the right side.
def preorder_traverse(self, focus_node):
```

```
        if focus_node is not None:
            print(focus_node)
            self.preorder_traverse(focus_node.left_child)
            self.preorder_traverse(focus_node.right_child)

    # With Post Order Traversal our Root is
    # going to come last. We start going to the
    # left child and then check both its children
    # The we'll check Roots Right Child and its children
    # finally checking Root last
    def postorder_traverse(self, focus_node):
        if focus_node is not None:
            self.postorder_traverse(focus_node.left_child)
            self.postorder_traverse(focus_node.right_child)
            print(focus_node)

    # In this function we will find Nodes
    def find(self, key):
        # Start at the top
        focus_node = self.root

        # While we haven't found it keep looking
        while focus_node.key != key:
            if key < focus_node.key:
                focus_node = focus_node.left_child
            else:
                focus_node = focus_node.right_child
            # If node wasn't found
            if focus_node is None:
                return None
        return focus_node


class Node:
    def __init__(self, key=0, name=""):
        self.key = key
        self.name = name
        # You can only have 1 left & right child
        self.left_child = None
        self.right_child = None

    def __str__(self):
        return f"{self.name} has the key {self.key}"


tree = BinaryTree()
tree.add_node(50, "Boss")
tree.add_node(25, "Vice President")
tree.add_node(15, "Office Manager")
tree.add_node(30, "Secretary")
tree.add_node(75, "Sales Manager")
tree.add_node(85, "Salesman")

# Test Different Ways to Traverse
```

```
# As we traverse the Nodes the keys increase in order
print(tree.in_order_traverse(tree.root))
print(tree.preorder_traverse(tree.root))
print(tree.postorder_traverse(tree.root))

# Find the salesman
print(tree.find(85))
# Search for someone that doesn't exist
print(tree.find(9))

# In the next video I'll show how to delete Nodes in trees
```

Data Structures & Algorithms 11

```
# In this video I'll focus on how you would delete nodes
# in a Binary Tree. This is the most complicated task
# you can perform with trees so refer to the tree diagrams
# while I cover the coding involved.

# Walkthrough Delete Root
# 1. Ask is the node we want to delete the Root?
# 2. If it is ask what is the right child of Root?
# 3. The right child will replace Root
# 4. Take the Roots left child and assign it as the left
# child to Roots right child

# Walkthrough Roots Left Child
# 1. Is the left child < Root? (We do this to determine if we
# are going to the left or right of Root)
# 2. The left child will be replaced by its right child
# 3. We do this by assigning 25s right child as the left child
# of Root
# 4. Then assign 25s left child as the left child of 30


class BinaryTree:
    def __init__(self):
        self.root = None

    def add_node(self, key, name):
        new_node = Node(key, name)
        if self.root is None:
            self.root = new_node
        else:
            focus_node = self.root
            while True:
                parent = focus_node
                if key < focus_node.key:
                    focus_node = focus_node.left_child
                    if focus_node is None:
                        parent.left_child = new_node
                        return True
                else:
```

```
                focus_node = focus_node.right_child
                if focus_node is None:
                    parent.right_child = new_node
                    return True

    def in_order_traverse(self, focus_node):
        if focus_node is not None:
            self.in_order_traverse(focus_node.left_child)
            print(focus_node)
            self.in_order_traverse(focus_node.right_child)

    # This function will remove nodes and adjust the tree
    def remove(self, key):
        # Start at the Root node
        focus_node = self.root

        # Also set parent as Root
        parent = self.root

        # Track are we moving to the left or right
        is_left_child = True

        # Search until we find it
        while focus_node.key != key:
            # Set parent as focus_node

            # Decide what direction to search in
            if key < focus_node.key:
                # We know we are searching to the left
                is_left_child = True
                # So set the focus node to the left child
                focus_node = focus_node.left_child
            else:
                # When it isn't a left child
                is_left_child = False
                # Set the focus as the right child
                focus_node = focus_node.right_child

            # If here we didn't find a match
            if focus_node is None:
                return False

        # If focus node doesn't have children and it is Root
        # set Root to None and we are done
        if focus_node.left_child is None and focus_node.right_child is None:
            if focus_node == self.root:
                self.root = None
            elif is_left_child:
                # If left child delete it by setting to None
                parent.left_child = None
            else:
                # Otherwise delete right child
                parent.right_child = None
        elif focus_node.right_child is None:
```

```
            # Handle if there is no right child
            if focus_node == self.root:
                # Move the left child into the root position
                root = focus_node.left_child
            elif is_left_child:
                # If left child set the parents left child
                parent.left_child = focus_node.left_child
            else:
                # Otherwise set to right child
                parent.right_child = focus_node.right_child
        elif focus_node.left_child is None:
            # Handle if there is no left child
            if focus_node == self.root:
                # Move the left child into the root position
                root = focus_node.right_child
            elif is_left_child:
                # If left child set the parents right child
                parent.left_child = focus_node.right_child
            else:
                # Otherwise set to left child
                parent.right_child = focus_node.left_child
        else:
            # 2 children are involved here and so I have to
            # figure out what the replacement should be
            replacement = self.get_replacement_node(focus_node)

            # If focus node is Root then I have to replace it
            if focus_node == self.root:
                root = replacement
            # Otherwise set either left or right child
            elif is_left_child:
                parent.left_child = replacement
            else:
                parent.right_child = replacement

            replacement.left_child = focus_node.left_child

            # If here it worked
            return True

    # Gets passed the node that should be replaced
    def get_replacement_node(self, replaced_node):
        # Parent of replaced node
        replacement_parent = replaced_node
        # The node that will replace the node passed here
        replacement = replaced_node
        # We always replace with the right child because its
        # value is bigger than the left child
        focus_node = replaced_node.right_child

        # While there are no more left children cycle until
        # the left child is moved into place [SLIDE]
        while focus_node is not None:
            replacement_parent = replacement
```

```
            replacement = focus_node
            focus_node = focus_node.left_child

        # If the replacement isn't the right child
        # move the replacement into the parents
        # left child slot and move the replaced nodes
        # right child into the replacements right child
        if replacement != replaced_node.right_child:
            replacement_parent.left_child = replacement.right_child
            replacement.right_child = replaced_node.right_child

        return replacement


class Node:
    def __init__(self, key=0, name=""):
        self.key = key
        self.name = name
        self.left_child = None
        self.right_child = None

    def __str__(self):
        return f"{self.name} has the key {self.key}"


tree = BinaryTree()
tree.add_node(50, "Boss")
tree.add_node(25, "Vice President")
tree.add_node(15, "Office Manager")
tree.add_node(30, "Secretary")
tree.add_node(75, "Sales Manager")
tree.add_node(85, "Salesman")

# Test replacing the Vice President
tree.in_order_traverse(tree.root)
tree.remove(25)
print("\nVice President Replaced\n")
tree.in_order_traverse(tree.root)
```

Data Structures & Algorithms 12

```
# What is a Heap?
# A heap is like a tree, but it is normally implemented
# as a list.
# With a heap every row must be complete. This means there
# must be a value in each node except for in the last row
# Parent nodes are larger than children, but unlike with a
# Binary Tree the left child isn't always bigger than the right
# Heaps can contain duplicates, and are fast at insertion,
# deletion and sorting.
# Heaps are slow when it comes to traversal & searching

# How Removal Works
```

```
# We pop off the top value and replace it with the lowest
# Then we percolate that value down as long as its value
# is greater than other values

# How Insertion Works
# Add the new value at the bottom and percolate up as long
# as its value is greater than others


class Node:
    def __init__(self, key):
        self.key = key

class Heap:
    def __init__(self, max_size):
        # Populate list with 31 Nones
        self.the_list = [None] * max_size
        self.max_size = max_size
        self.current_size = 0

    # Tests if list is empty
    def is_empty(self):
        return self.current_size == 0

    # Will insert a new Node using the provided key
    def insert(self, key):
        # Make sure list isn't full
        if self.current_size == self.max_size:
            return False

        # Create new node for the list
        new_node = Node(key)

        # Assign node after last assigned value
        self.the_list[self.current_size] = new_node

        # Keep track of the number of items in list
        self.current_size += 1

        # Pass the index for the new value which will be positioned
        self.percolate_up(self.current_size-1)
        return True

    def percolate_up(self, index):
        # Get the new nodes parent
        parent = int((index - 1) / 2)

        # The new node added
        bottom = self.the_list[index]

        # If not at top of list and new node is greater
        while index > 0 and self.the_list[parent].key < bottom.key:
            # Move new node up and prepare to test the next parent
            self.the_list[index] = self.the_list[parent]
```

```
            index = parent
            parent = int((parent - 1) / 2)

        # Assign current index with the new node
        self.the_list[index] = bottom

    # Remove max value
    def pop(self):
        # Get the root
        root = self.the_list[0]
        # Decrement to new list size
        self.current_size -= 1
        # Move bottom node to top
        self.the_list[0] = self.the_list[self.current_size]
        # Move all nodes into position starting at the top
        self.percolate_down(0)
        return root

    # This function moves values into position starting at the top
    def percolate_down(self, index):
        # Will hold the larger of the children nodes
        larger_child = 0

        # Gets the top node in the list
        top = self.the_list[index]

        # We don't have to check the bottom row
        while index < self.current_size / 2:
            # Gets the index for the left & right child
            left_child = 2 * index + 1
            right_child = left_child + 1

            # Avoid None valued Nodes and if left child is < right
            if right_child < self.current_size and self.the_list[left_child].key <
self.the_list[right_child].key:

                # Then save the right child index as the largest
                larger_child = right_child
            else:
                # Otherwise the left child is the largest
                larger_child = left_child

            # If the top value is ever greater than the largest
            # child jump out of the while
            if top.key >= self.the_list[larger_child].key:
                break

            # Assign the largest node
            self.the_list[index] = self.the_list[larger_child]

            # Set the index to that largest node
            index = larger_child

        # After we finish cycling assign the top value
```

```
        self.the_list[index] = top


heap = Heap(31)
heap.insert(72)
heap.insert(44)
heap.insert(53)
heap.insert(21)
heap.insert(66)
heap.insert(100)
heap.insert(84)
heap.insert(35)
heap.insert(19)
heap.insert(90)

# Pop off the 100
heap.pop()

for i in heap.the_list:
    if i is None:
        print("N", end=", ")
    else:
        print(i.key, end=", ")
print()
```