

```
# Programming involves listing all the things that must happen to solve a problem
# When writing a program first determine step-by-step what needs to happen
# Then convert those steps into the language being Python in this situation
```

```
# Every language has the following
# 1. The ability to accept input and store it in many ways
# 2. The ability to output information to the screen, files, etc.
# 3. The ability to conditionally do one thing or another thing
# 4. The ability to do something multiple times
# 5. The ability to make mathematical calculations
# 6. The ability to change data
# 7. (Object Oriented Programming) Model real world objects using code
```

```
# ----- Hello World -----
```

```
# Ask the user to input their name and assign it to the name variable
# Variable names start with a letter or _ and then can contain numbers
# Names are case sensitive so name is not the same as Name
name = input('What is your name ')
```

```
# Print out Hello followed by the name they entered
print('Hello ', name)
```

```
# You can't use the following 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
```

```
# Single line comments are ignored by the interpreter
'''
So are multiline comments
'''
```

```
# ----- MATH ON 2 VALUES -----
```

```
# Ask the user to input 2 values and store them in variables num1 and num2
# split() splits input using whitespace
num1, num2 = input('Enter 2 numbers : ').split()
```

```
# Convert strings into regular numbers (integers)
num1 = int(num1)
num2 = int(num2)
```

```
# Add the values entered and store in sum
sum = num1 + num2
```

```
# Subtract the values and store in difference
difference = num1 - num2
```

```
# Multiply the values and store in product
product = num1 * num2
```

```
# Divide the values and store in quotient
quotient = num1 / num2
```

```
# Use modulus on the values to find the remainder
remainder = num1 % num2
```

```
# Print your results
# format() loads the variable values in order into the {} placeholders
print("{} + {} = {}".format(num1, num2, sum))
print("{} - {} = {}".format(num1, num2, difference))
print("{} * {} = {}".format(num1, num2, product))
print("{} / {} = {}".format(num1, num2, quotient))
print("{} % {} = {}".format(num1, num2, remainder))
```

```
# ----- PROBLEM : MILES TO KILOMETERS -----
# Sample knowing that kilometers = miles * 1.60934
# Enter Miles 5
# 5 miles equals 8.0467 kilometers
```

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

```
miles = input('Enter Miles ')\n\n# Convert from string to integer\nmiles = int(miles)\n\n# Perform calculation by multiplying 1.60934 times miles\nkilometers = miles * 1.60934\n\n# Print results using format()\nprint("{} miles equals {} kilometers".format(miles, kilometers))\n\n# ----- CALCULATOR ----- \n# Receive 2 numbers separated by an operator and show a result\n# Sample\n# Enter Calculation: 5 * 6\n# 5 * 6 = 30\n\n# Store the user input of 2 numbers and an operator\nnum1, operator, num2 = input('Enter Calculation: ').split()\n\n# Convert strings into integers\nnum1 = int(num1)\nnum2 = int(num2)\n\n# If, else if (elif) and else execute different code depending on a condition\nif operator == "+":\n    print("{} + {} = {}".format(num1, num2, num1+num2))\n\n# If the 1st condition wasn't true check if this one is\nelif operator == "-":\n    print("{} - {} = {}".format(num1, num2, num1 - num2))\nelif operator == "*":\n    print("{} * {} = {}".format(num1, num2, num1 * num2))\nelif operator == "/":\n    print("{} / {} = {}".format(num1, num2, num1 / num2))\n\n# If none of the above conditions were true then execute this by default\nelse:\n    print("Use either + - * or / next time")\n\n# Other conditional operators\n# > : Greater than\n# < : Less than\n# >= : Greater than or equal to\n# <= : Less than or equal to\n# != : Not equal to\n\n# ----- IS BIRTHDAY IMPORTANT ----- \n# We'll provide different output based on age\n# 1 - 18 -> Important\n# 21, 50, > 65 -> Important\n# All others -> Not Important\n\n# eval() converts a string into an integer if it meets the guidelines\nage = eval(input("Enter age: "))\n\n# Logical operators can be used to combine conditions\n# and : If both are true it returns true\n# or : If either are true it returns true\n# not : Converts true into false and vice versa\n\n# If age is both greater than or equal to 1 and less than or equal to 18 it is true\nif (age >= 1) and (age <= 18):\n    print("Important Birthday")\n\n# If age is either 21 or 50 then it is true\nelif (age == 21) or (age == 50):\n    print("Important Birthday")\n\n# We check if age is less than 65 and then convert true to false or vice versa\n# This is the same as if we put age > 65\nelif not(age < 65):\n    print("Important Birthday")\nelse:
```

```
print("Sorry Not Important")

# ----- PROBLEM : DETERMINE GRADE -----
# If age 5 go to kindergarten
# Ages 6 through 17 goes to grades 1 through 12
# If age is greater then 17 then say go to college
# Try to complete with 10 or less lines

# Ask for the age
age = eval(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")
```

```
Learn to program 2
2 # values. The for loop can be used to do this.
3
4 # Each time we go through the loop variable i's value will be
5 # assigned the next value in our list
6
7 for i in [2,4,6,8,10]:
8     print("i = ", i)
9
10 # We can also have range define our list for us. range(10) will
11 # create a list starting at 0 and go up to but not include
12 # the value passed to it.
13
14 for i in range(10):
15     print("i = ", i)
16
17 # We can define the starting and ending value for range
18 for i in range(2, 10):
19     print("i = ", i)
20
21 # You can use modulus to see if a number is odd or even
22 # If we divide an even by 2 there will be no remainder
23 # so if i % 2 == 0 we know it is true
24 i = 2
25 print((i % 2) == 0)
26
27 # ----- PROBLEM : PRINT ODDS FROM 1 to 20 -----
28 # Use a for loop, range, if and modulus to print out the odds
29
30 # Use for to loop through the list from 1 to 21
31
32 for i in range(1, 21):
33
34     # Use modulus to check that the result is NOT EQUAL to 0
35     # Print the odds
36
37     if ((i % 2) != 0):
38         print("i = ", i)
39
40 # ----- WORKING WITH FLOATS -----
41 # Floating point numbers are numbers with decimal values
42
43 your_float = input("Enter a float: ")
44
45 # We can convert a string into a float like this
46
47 your_float = float(your_float)
48
49 # We can define how many decimals are printed being 2
50 # here by putting :.2 before f
51 print("Rounded to 2 decimals : {:.2f}".format(your_float))
52
53 # ----- PROBLEM : COMPOUNDING INTEREST -----
54 # Have the user enter their investment amount and expected interest
55 # Each year their investment will increase by their investment +
56 # their investment * the interest rate
57 # Print out their earnings after a 10 year period
58
59 # Ask for money invested + the interest rate
60 money = input("How much to invest: ")
61 interest_rate = input("Interest Rate: ")
62
63 # Convert value to a float
64 money = float(money)
65
66 # Convert value to a float and round the percentage rate by 2 digits
67 interest_rate = float(interest_rate) * .01
68
69 # Cycle through 10 years using for and range from 0 to 9
```

138

```
139     # If i equals 15 stop looping
140     if i == 15:
141         break
142
143     # Print the odds
144     print("Odd : ", i)
145
146     # Increment i
147     i += 1
148
149 # ----- PROBLEM : DRAW A PINE TREE -----
150 # For this problem I want you to draw a pine tree after asking the user
151 # for the number of rows. Here is the sample program
152
153 '''
154 How tall is the tree : 5
155     #
156     ###
157     #####
158     #######
159     #########
160     #
161 '''
162
163 # You should use a while loop and 3 for loops
164
165 # I know that this is the number of spaces and hashes for the tree
166 # 4 - 1
167 # 3 - 3
168 # 2 - 5
169 # 1 - 7
170 # 0 - 9
171 # Spaces before stump = Spaces before top
172
173 # So I need to
174 # 1. Decrement spaces by one each time through the loop
175 # 2. Increment the hashes by 2 each time through the loop
176 # 3. Save spaces to the stump by calculating tree height - 1
177 # 4. Decrement from tree height until it equals 0
178 # 5. Print spaces and then hashes for each row
179 # 6. Print stump spaces and then 1 hash
180
181 # Get the number of rows for the tree
182 tree_height = input("How tall is the tree : ")
183
184 # Convert into an integer
185 tree_height = int(tree_height)
186
187 # Get the starting spaces for the top of the tree
188 spaces = tree_height - 1
189
190 # There is one hash to start that will be incremented
191 hashes = 1
192
193 # Save stump spaces til later
194 stump_spaces = tree_height - 1
195
196 # Makes sure the right number of rows are printed
197 while tree_height != 0:
198
199     # Print the spaces
200     # end="" means a newline won't be added
201     for i in range(spaces):
202         print(' ', end="")
203
204     # Print the hashes
205     for i in range(hashes):
206         print('#', end="")
207
```

```
208 # Newline after each row is printed
209 print()
210
211 # I know from research that spaces is decremented by 1 each time
212 spaces -= 1
213
214 # I know from research that hashes is incremented by 2 each time
215 hashes += 2
216
217 # Decrement tree height each time to jump out of loop
218 tree_height -= 1
219
220 # Print the spaces before the stump and then a hash
221 for i in range(stump_spaces):
222     print(' ', end="")
223
224 print("#")
```

```
1 # ----- FORCE USER TO ENTER A NUMBER -----
2 # By giving the while a value of True it will cycle until a break is reached
3 while True:
4
5     # If we expect an error can occur surround potential error with try
6     try:
7         number = int(input("Please enter a number : "))
8         break
9
10    # The code in the except block provides an error message to set things right
11    # We can either target a specific error like ValueError
12    except ValueError:
13        print("You didn't enter a number")
14
15    # We can target all other errors with a default
16    except:
17        print("An unknown error occurred")
18
19 print("Thank you for entering a number")
20
21 # ----- Problem : Implement a Do While Loop -----
22 # Now I want you to implement a Do While loop in Python
23 # They always execute all of the code at least once and at
24 # the end check if a condition is true that would warrant
25 # running the code again. They have this format
26 # do {
27 #     ... Bunch of code ...
28 # } while(condition)
29
30 # I'll create a guessing game in which the user must chose
31 # a number between 1 and 10 of the following format
32 '''
33 Guess a number between 1 and 10 : 1
34 Guess a number between 1 and 10 : 3
35 Guess a number between 1 and 10 : 5
36 Guess a number between 1 and 10 : 7
37 You guessed it
38 '''
39
40 # Hint : You'll need a while and break
41
42 secret_number = 7
43
44 while True:
45     guess = int(input("Guess a number between 1 and 10 : "))
46
47     if guess == secret_number:
48         print("You guessed it")
49         break
50
51 # ----- MATH MODULE -----
52 # Python provides many functions with its Math module
53 import math
54
55 # Because you used import you access methods by referencing the module
56 print("ceil(4.4) = ", math.ceil(4.4))
57 print("floor(4.4) = ", math.floor(4.4))
58 print("fabs(-4.4) = ", math.fabs(-4.4))
59
60 # Factorial = 1 * 2 * 3 * 4
61 print("factorial(4) = ", math.factorial(4))
62
63 # Return remainder of division
64 print("fmod(5,4) = ", math.fmod(5,4))
65
66 # Receive a float and return an int
67 print("trunc(4.2) = ", math.trunc(4.2))
68
69 # Return x^y
```



```
70 print("pow(2,2) = ", math.pow(2,2))
71
72 # Return the square root
73 print("sqrt(4) = ", math.sqrt(4))
74
75 # Special values
76 print("math.e = ", math.e)
77 print("math.pi = ", math.pi)
78
79 # Return e^x
80 print("exp(4) = ", math.factorial(4))
81
82 # Return the natural logarithm e * e * e ~= 20 so log(20) tells
83 # you that e^3 ~= 20
84 print("log(20) = ", math.log(20))
85
86 # You can define the base and 10^3 = 1000
87 print("log(1000,10) = ", math.log(1000,10))
88
89 # You can also use base 10 like this
90 print("log10(1000) = ", math.log10(1000))
91
92 # We have the following trig functions
93 # sin, cos, tan, asin, acos, atan, atan2, asinh, acosh,
94 # atanh, sinh, cosh, tanh
95
96 # Convert radians to degrees and vice versa
97 print("degrees(1.5708) = ", math.degrees(1.5708))
98 print("radians(90) = ", math.radians(90))
99
100 # ----- ACCURATE FLOATING POINT CALCULATIONS -----
101
102 # The decimal module provides for more accurate floating point calculations
103 # With from you can reference methods without the need to reference the module
104 # like we had to do with math above
105 # We create an alias being D here to avoid conflicts with methods with
106 # the same name
107 from decimal import Decimal as D
108
109 sum = D(0)
110 sum += D("0.01")
111 sum += D("0.01")
112 sum += D("0.01")
113 sum -= D("0.03")
114
115 print("Sum = ", sum)
116
117 # ----- STRINGS -----
118 # Text is stored using the string data type
119 # You can use type to see the data type of a value
120 print(type(3))
121 print(type(3.14))
122
123 # Single quotes or double are both used for strings
124 print(type("3"))
125 print(type('3'))
126
127 samp_string = "This is a very important string"
128
129 # Each character is stored in a series of boxes labeled with index numbers
130 # You can get a character by referencing an index
131 print(samp_string[0])
132
133 # Get the last character
134 print(samp_string[-1])
135
136 # Get the last character
137 print(samp_string[3+5])
138
```

```
139 # Get the string length
140 print("Length :", len(samp_string))
141
142 # Get a slice by saying where to start and end
143 # The 4th index isn't returned
144 print(samp_string[0:4])
145
146 # Get everything starting at an index
147 print(samp_string[8:])
148
149 # Join or concatenate strings with +
150 print("Green " + "Eggs")
151
152 # Repeat strings with *
153 print("Hello " * 5)
154
155 # Convert an int into a string
156 num_string = str(4)
157
158 # Cycle through each character with for
159 for char in samp_string:
160     print(char)
161
162 # Cycle through characters in pairs
163 # Subtract 1 from the length because length is 1 more then the highest index
164 # because strings are 0 indexed
165 # Use range starting at index 0 through string length and increment by
166 # 2 each time through
167 for i in range(0, len(samp_string)-1, 2):
168     print(samp_string[i] + samp_string[i+1])
169
170 # Computers assign characters with a number known as a Unicode
171 # A-Z have the numbers 65-90 and a-z 97-122
172
173 # You can get the Unicode code with ord()
174 print("A =", ord("A"))
175
176 # You can convert from Unicode with chr
177 print("65 =", chr(65))
178
179 # ----- PROBLEM : SECRET STRING -----
180 # Receive a uppercase string and then hide its meaning by turning
181 # it into a string of unicodes
182 # Then translate it from unicode back into its original meaning
183
184 norm_string = input("Enter a string to hide in uppercase: ")
185
186 secret_string = ""
187
188 # Cycle through each character in the string
189 for char in norm_string:
190
191     # Store each character code in a new string
192     secret_string += str(ord(char))
193
194 print("Secret Message :", secret_string)
195
196 norm_string = ""
197
198 # Cycle through each character code 2 at a time by incrementing by
199 # 2 each time through since unicodes go from 65 to 90
200 for i in range(0, len(secret_string)-1, 2):
201
202     # Get the 1st and 2nd for the 2 digit number
203     char_code = secret_string[i] + secret_string[i+1]
204
205     # Convert the codes into characters and add them to the new string
206     norm_string += chr(int(char_code))
207
```

```
208 print("Original Message :", norm_string)
209
210 # ----- SUPER AWESOME MIND SCRAMBLING PROBLEM -----
211 # Make the above work with upper and lowercase with 1 addition and
212 # 1 subtraction
213
214 # Add these 2 changes
215
216 # secret_string += str(ord(char) - 23)
217
218 # norm_string += chr(int(char_code) + 23)
```

If you like videos like this consider [donating a dollar on Patreon](#).

Code & Transcript

```
Learn to program 4
2
3 # Strings have many methods we can use beyond what I covered last time
4 rand_string = "   this is an important string   "
5
6 # Delete whitespace on left
7 rand_string = rand_string.lstrip()
8
9 # Delete whitespace on right
10 rand_string = rand_string.rstrip()
11
12 # Delete whitespace on right and left
13 rand_string = rand_string.strip()
14
15 # Capitalize the 1st letter
16 print(rand_string.capitalize())
17
18 # Capitalize every letter
19 print(rand_string.upper())
20
21 # lowercase all letters
22 print(rand_string.lower())
23
24 # Turn a list into a string and separate items with the defined
25 # separator
26 a_list = ["Bunch", "of", "random", "words"]
27 print(" ".join(a_list))
28
29 # Convert a string into a list
30 a_list_2 = rand_string.split()
31
32 for i in a_list_2:
33     print(i)
34
35 # Count how many times a string occurs in a string
36 print("How many is :", rand_string.count("is"))
37
38 # Get index of matching string
39 print("Where is string :", rand_string.find("string"))
40
41 # Replace a substring
42 print(rand_string.replace("an ", "a kind of "))
43
44 # ----- PROBLEM : ACRONYM GENERATOR -----
45 # You will enter a string and then convert it to an acronym
46 # with uppercase letters like this
47 '''
48 Convert to Acronym : Random Access Memory
49 RAM
50 '''
51
52 # Ask for a string
53 orig_string = input("Convert to Acronym : ")
54
55 # Convert the string to all uppercase
56 orig_string = orig_string.upper()
57
58 # Convert the string into a list
59 list_of_words = orig_string.split()
60
61 # Cycle through the list
62 for word in list_of_words:
63
```

```
64 # Get the 1st letter of the word and eliminate the newline
65 print(word[0], end="")
66
67 print()
68
69 # ----- MORE STRING METHODS -----
70 # For our next problem some additional string methods are going to be
71 # very useful
72
73 letter_z = "z"
74 num_3 = "3"
75 a_space = " "
76
77 # Returns True if characters are letters or numbers
78 # Whitespace is false
79 print("Is z a letter or number :", letter_z.isalnum())
80
81 # Returns True if characters are letters
82 print("Is z a letter :", letter_z.isalpha())
83
84 # Returns True if characters are numbers (Floats are False)
85 print("Is 3 a number :", num_3.isdigit())
86
87 # Returns True if all are lowercase
88 print("Is z a lowercase :", letter_z.islower())
89
90 # Returns True if all are uppercase
91 print("Is z a uppercase :", letter_z.isupper())
92
93 # Returns True if all are spaces
94 print("Is space a space :", a_space.isspace())
95
96 # ----- MAKE A isfloat FUNCTION -----
97 # There is no way to check if a string contains a float
98 # so let's make one by defining our own function
99
100 # Functions allow use to avoid repeating code
101 # They can receive values (attributes / parameters)
102 # They can return values
103
104 # You define your function name and the names for the values
105 # it receives like this
106
107 def isfloat(str_val):
108     try:
109
110         # If the string isn't a float float() will throw a
111         # ValueError
112         float(str_val)
113
114         # If there is a value you want to return use return
115         return True
116     except ValueError:
117         return False
118
119 pi = 3.14
120
121 # We call our functions by name and pass in a value between
122 # the parentheses
123 print("Is Pi a Float :", isfloat(pi))
124
125 # ----- PROBLEM : CAESAR'S CIPHER -----
126 # Receive a message and then encrypt it by shifting the
127 # characters by a requested amount to the right
128 # A becomes D, B becomes E for example
129 # Also decrypt the message back again
130
131 # A-Z have the numbers 65-90 in unicode
132 # a-z have the numbers 97-122
```

```
133 # You get the unicode of a character with ord(yourLetter)
134 # You convert from unicode to character with chr(yourNumber)
135
136 # You should check if a character is a letter and if not
137 # leave it as its default
138
139 # Hints
140 # Use isupper() to decided which unicodes to work with
141 # Add the key (number of characters to shift) and if
142 # bigger or smaller then the unicode for A, Z, a, or z
143 # increase or decrease by 26
144
145 # Receive the message to encrypt and the number of characters to shift
146 message = input("Enter your message : ")
147 key = int(input("How many characters should we shift (1 - 26)"))
148
149 # Prepare your secret message
150 secret_message = ""
151
152 # Cycle through each character in the message
153 for char in message:
154
155     # If it isn't a letter then keep it as it is in the else below
156     if char.isalpha():
157
158         # Get the character code and add the shift amount
159         char_code = ord(char)
160         char_code += key
161
162         # If uppercase then compare to uppercase unicodes
163         if char.isupper():
164
165             # If bigger than Z subtract 26
166             if char_code > ord('Z'):
167                 char_code -= 26
168
169             # If smaller than A add 26
170             elif char_code < ord('A'):
171                 char_code += 26
172
173         # Do the same for lowercase characters
174         else:
175             if char_code > ord('z'):
176                 char_code -= 26
177             elif char_code < ord('a'):
178                 char_code += 26
179
180         # Convert from code to letter and add to message
181         secret_message += chr(char_code)
182
183     # If not a letter leave the character as is
184     else:
185         secret_message += char
186
187 print("Encrypted :", secret_message)
188
189 # To decrypt the only thing that changes is the sign of the key
190 key = -key
191
192 orig_message = ""
193
194 for char in secret_message:
195     if char.isalpha():
196         char_code = ord(char)
197         char_code += key
198
199         if char.isupper():
200             if char_code > ord('Z'):
201                 char_code -= 26
```

```
202         elif char_code < ord('A'):  
203             char_code += 26  
204     else:  
205         if char_code > ord('z'):  
206             char_code -= 26  
207         elif char_code < ord('a'):  
208             char_code += 26  
209  
210         orig_message += chr(char_code)  
211  
212     else:  
213         orig_message += char  
214  
215 print("Decrypted :", orig_message)  
216  
217 # ----- EXTRA HOMEWORK -----  
218 # For homework put the duplicate code above in a function
```

```
1  # ----- FUNCTION BASICS -----
2
3  # Functions allow use to reuse code and make the code easier
4  # to understand
5
6  # To create a function type def (define) the function name
7  # and then in parentheses a comma separated list of values
8  # to pass if needed
9
10 def add_numbers(num_1, num2):
11
12     # Return returns a value if needed
13     return num_1 + num2
14
15 # You call the function by name followed by passing comma
16 # separated values if needed and a value may or may not be
17 # returned
18
19 print("5 + 4 =", add_numbers(5, 4))
20
21 # ----- FUNCTION LOCAL VARIABLES -----
22 # Any variable defined inside of a function is available only
23 # in that function
24
25 # ----- EXAMPLE 1 -----
26 # Variables created in a function can't be accessed outside
27 # of it
28
29 def assign_name():
30     name = "Doug"
31
32 assign_name()
33
34 # Throws a NameError
35 # print(name)
36
37 # ----- EXAMPLE 2 -----
38
39 # You can't change a global variable even if it is passed
40 # into a function
41 def change_name(name):
42
43     # Trying to change the global
44     name = "Mark"
45
46 # A variable defined outside of a function can't be changed
47 # in the function using the above way
48 name = "Tom"
49
50 # Try to change the value
51 change_name(name)
52
53 # Prints Tom even though the function tries to change it
54 print(name)
55
56 # ----- EXAMPLE 3 -----
57
58 # If you want to change the value pass it back
59 def change_name_2():
60     return "Mark"
61
62 name = change_name_2()
63
64 print(name)
65
66 # ----- EXAMPLE 4 -----
67 # You can also use the global statement to change it
68
69 gbl_name = "Sally"
```



```
70
71 def change_name_3():
72     global gbl_name
73     gbl_name = "Sammy"
74
75 change_name_3()
76
77 print(gbl_name)
78
79 # ----- RETURNING NONE -----
80 # If you don't return a value a function will return None
81
82 def get_sum(num1, num2):
83     sum = num1 + num2
84
85 print(get_sum(5, 4))
86
87 # ----- PROBLEM : SOLVE FOR X -----
88 # Make a function that receives an algebraic equation like
89 # x + 4 = 9 and solve for x
90 # x will always be the 1st value received and you only
91 # will deal with addition
92
93 # Receive the string and split the string into variables
94 def solve_eq(equation):
95     x, add, num1, equal, num2 = equation.split()
96
97     # Convert the strings into ints
98     num1, num2 = int(num1), int(num2)
99
100    # Convert the result into a string and join (concatenate)
101    # it to the string "x = "
102    return "x = " + str(num2 - num1)
103
104 print(solve_eq("x + 4 = 9"))
105
106 # ----- RETURN MULTIPLE VALUES -----
107 # You can return multiple values with the return statement
108
109 def mult_divide(num1, num2):
110     return (num1 * num2), (num1 / num2)
111
112 mult, divide = mult_divide(5, 4)
113
114 print("5 * 4 =", mult)
115 print("5 / 4 =", divide)
116
117 # ----- RETURN A LIST OF PRIMES -----
118 # A prime can only be divided by 1 and itself
119 # 5 is prime because 1 and 5 are its only positive factors
120 # 6 is a composite because it is divisible by 1, 2, 3 and 6
121
122 # We'll receive a request for primes up to the input value
123 # We'll then use a for loop and check if modulus == 0 for
124 # every value up to the number to check
125 # If modulus == 0 that means the number isn't prime
126
127 def isprime(num):
128     # This for loop cycles through primes from 2 to
129     # the value to check
130     for i in range(2, num):
131
132         # If any division has no remainder we know it
133         # isn't a prime number
134         if (num % i) == 0:
135             return False
136     return True
137
138
```

```
139 def getPrimes(max_number):
140
141     # Create a list to hold primes
142     list_of_primes = []
143
144     # This for loop cycles through primes from 2 to
145     # the maximum value requested
146     for num1 in range(2, max_number):
147
148         if isprime(num1):
149             list_of_primes.append(num1)
150
151     return list_of_primes
152
153 max_num_to_check = int(input("Search for Primes up to : "))
154
155 list_of_primes = getPrimes(max_num_to_check)
156
157 for prime in list_of_primes:
158     print(prime)
159
160 # ----- UNKNOWN NUMBER OF ARGUMENTS -----
161 # We can receive an unknown number of arguments using
162 # the splat (*) operator
163
164 def sumAll(*args):
165
166     sum = 0
167
168     for i in args:
169         sum += i
170
171     return sum
172
173 print("Sum :", sumAll(1,2,3,4))
174
175 # ----- pythontut2.py -----
176
177 # We need this module for our program
178 import math
179
180 # Functions allow us to avoid duplicate code in our programs
181
182 # Aside from having to type code twice duplicate code is bad
183 # because it requires us to change multiple blocks of code
184 # if we need to make a change
185
186 # ----- OUR FUNCTIONS -----
187
188 # This routes to the correct area function
189 # The name of the value passed doesn't have to match
190 def get_area(shape):
191
192     # Switch to lowercase for easy comparison
193     shape = shape.lower()
194
195     if shape == "rectangle":
196         rectangle_area()
197     elif shape == "circle":
198         circle_area()
199     else:
200         print("Please enter rectangle or circle")
201
202 # Create function that calculates the rectangle area
203 def rectangle_area():
204     length = float(input("Enter the length : "))
205     width = float(input("Enter the width : "))
206
207     area = length * width
```

```
208
209     print("The area of the rectangle is", area)
210
211
212 # Create function that calculates the circle area
213 def circle_area():
214     radius = float(input("Enter the radius : "))
215
216     area = math.pi * (math.pow(radius, 2))
217
218     # Format the output to 2 decimal places
219     print("The area of the circle is {:.2f}".format(area))
220
221
222 # ----- END OF OUR FUNCTIONS -----
223
224 # We often place our main programming logic in a function called main
225 # We create it this way
226
227 def main():
228
229     # Our program will calculate the area for rectangles or circles
230
231     # Without functions we'd have to create a giant list of ifs, elifs
232
233     # Ask the user what shape they have
234     shape_type = input("Get area for what shape : ")
235
236     # Call a function that will route to the correct function
237     get_area(shape_type)
238
239     # Because of functions it is very easy to see what is happening
240     # For more detail just refer to the very short specific functions
241
242 # All of the function definitions are ignored and this calls for main()
243 # to execute when the program starts
244
245 main()
246
247 # ----- HOMEWORK -----
248 # Add the ability to calculate the area for parallelograms,
249 # rhombus, triangles, and trapezoids
```

```
# ----- LEARN TO PROGRAM 6 -----
3
4 import random
5 import math
6
7 # With lists we can refer to groups of data with 1 name
8
9 # Each item in the list corresponds to a number (index)
10 # just like how people have identification numbers.
11 # By default the 1st item in a list has the index 0
12
13 # [0 : "string"] [1 : 1.234] [2 : 28] [3 : "c"]
14
15 # Python lists can grow in size and can contain data
16 # of any type
17
18 randList = ["string", 1.234, 28]
19
20 # Create a list with range
21 oneToTen = list(range(10))
22
23 # An awesome thing about lists is that you can use many
24 # of the same functions with them that you used with strings
25
26 # Combine lists
27 randList = randList + oneToTen
28
29 # Get the 1st item with an index
30 print(randList[0])
31
32 # Get the length
33 print("List Length :", len(randList))
34
35 # Slice a list to get 1st 3 items
36 first3 = randList[0:3]
37
38 # Cycle through the list and print the index
39 for i in first3:
40     print("{} : {}".format(first3.index(i), i))
41
42 # You can repeat a list item with *
43 print(first3[0] * 3)
44
45 # You can see if a list contains an item
46 print("string" in first3)
47
48 # You can get the index of a matching item
49 print("Index of string :", first3.index("string"))
50
51 # Find out how many times an item is in the list
52 print("How many strings :", first3.count("string"))
53
54 # You can change a list item
55 first3[0] = "New String"
56
57 for i in first3:
58     print("{} : {}".format(first3.index(i), i))
59
60 # Append a value to the end of a list
61 first3.append("Another")
62
63 # ----- PROBLEM : CREATE A RANDOM LIST -----
64 # Generate a random list of 5 values between 1 and 9
65 numList = []
66 for i in range(5):
67     numList.append(random.randrange(1, 9))
68
69 # ----- SORT A LIST : BUBBLE SORT -----
70 # The Bubble sort is a way to sort a list
```

```
71 # It works this way
72 # 1. An outer loop decreases in size each time
73 # 2. The goal is to have the largest number at the end of
74 #    the list when the outer loop completes 1 cycle
75 # 3. The inner loop starts comparing indexes at the beginning
76 #    of the loop
77 # 4. Check if list[Index] > list[Index + 1]
78 # 5. If so swap the index values
79 # 6. When the inner loop completes the largest number is at
80 #    the end of the list
81 # 7. Decrement the outer loop by 1
82
83 # Create the value that will decrement for the outer loop
84 # Its value is the last index in the list
85 i = len(numList) - 1
86
87 while i > 1:
88     j = 0
89
90     while j < i:
91
92         # Tracks the comparison of index values
93         print("\nIs {} > {}".format(numList[j], numList[j+1]))
94         print()
95
96         # If the value on the left is bigger switch values
97         if numList[j] > numList[j+1]:
98
99             print("Switch")
100
101             temp = numList[j]
102             numList[j] = numList[j + 1]
103             numList[j + 1] = temp
104
105         else:
106             print("Don't Switch")
107
108         j += 1
109
110     # Track changes to the list
111     for k in numList:
112         print(k, end=" ", " ")
113     print()
114
115     print("END OF ROUND")
116
117     i -= 1
118
119 for k in numList:
120     print(k, end=" ", " ")
121 print()
122
123 # ----- MORE LIST FUNCTIONS -----
124 numList = []
125 for i in range(5):
126     numList.append(random.randrange(1, 9))
127
128 # Sort a list
129 numList.sort()
130
131 # Reverse a list
132 numList.reverse()
133
134 for k in numList:
135     print(k, end=" ", " ")
136 print()
137
138 # Insert value at index insert(index, value)
```

```
140 numList.insert(5, 10)
141
142 # Delete first occurrence of value
143 numList.remove(10)
144
145 for k in numList:
146     print(k, end=", ")
147 print()
148
149 # Remove item at index
150 numList.pop(2)
151
152 for k in numList:
153     print(k, end=", ")
154 print()
155
156 # ----- LIST COMPREHENSIONS -----
157 # You can construct lists in interesting ways using
158 # list comprehensions
159
160 # Perform an operation on each item in the list
161
162 # Create a list of even values
163 evenList = [i*2 for i in range(10)]
164
165 for k in evenList:
166     print(k, end=", ")
167 print()
168
169 # List of lists containing values to the power of
170 # 2, 3, 4
171 numList = [1,2,3,4,5]
172
173 listOfValues = [[math.pow(m, 2), math.pow(m, 3), math.pow(m, 4)]
174                 for m in numList]
175
176 for k in listOfValues:
177     print(k)
178 print()
179
180 # Create a 10 x 10 list
181 multiDList = [[0] * 10 for i in range(10)]
182
183 # Change a value in the multidimensional list
184 multiDList[0][1] = 10
185
186 # Get the 2nd item in the 1st list
187 # It may help to think of it as the 2nd item in the 1st row
188 print(multiDList[0][1])
189
190 # Get the 2nd item in the 2nd list
191 print(multiDList[1][1])
192
193 # ----- MULTIDIMENSIONAL LIST -----
194
195 # Show how indexes work with a multidimensional list
196 listTable = [[0] * 10 for i in range(10)]
197
198 for i in range(10):
199     for j in range(10):
200         listTable[i][j] = "{} : {}".format(i, j)
201
202 for i in range(10):
203     for j in range(10):
204         print(listTable[i][j], end=" || ")
205     print()
206
207
208
```

```
209 # ----- PROBLEM : CREATE MULTIPLICATION TABLE -----
210 # With 2 for loops fill the cells in a multidimensional
211 # list with a multiplication table using values 1 - 9
212 '''
213 1, 2, 3, 4, 5, 6, 7, 8, 9,
214 2, 4, 6, 8, 10, 12, 14, 16, 18,
215 3, 6, 9, 12, 15, 18, 21, 24, 27,
216 4, 8, 12, 16, 20, 24, 28, 32, 36,
217 5, 10, 15, 20, 25, 30, 35, 40, 45,
218 6, 12, 18, 24, 30, 36, 42, 48, 54,
219 7, 14, 21, 28, 35, 42, 49, 56, 63,
220 8, 16, 24, 32, 40, 48, 56, 64, 72,
221 9, 18, 27, 36, 45, 54, 63, 72, 81
222 '''
223
224 # Create the multidimensional list
225 multTable = [[0] * 10 for i in range(10)]
226
227 # This will increment for each row
228 for i in range(1, 10):
229
230     # This will increment for each item in the row
231     for j in range(1, 10):
232
233         # Assign the value to the cell
234         multTable[i][j] = i * j
235
236 # Output the data in the same way you assigned it
237 for i in range(1, 10):
238
239     for j in range(1, 10):
240         print(multTable[i][j], end=", ")
241
242     print()
```

```

# ----- 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

# Create a Dictionary about me
derekDict = {"fName": "Derek", "lName": "Banas", "address": "123 Main St"}

# Get a value with the key
print("May name :", derekDict["fName"])

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

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

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

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

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

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

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

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

# Delete a key value
del derekDict["fName"]

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

# Delete all entries
derekDict.clear()

# List for holding Dictionaries
employees = []

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

employees.append({'fName': fName, 'lName': lName})

print(employees)

# ----- PROBLEM : CREATE A CUSTOMER LIST -----
# Create an array of customer dictionaries
# 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
'''

```



```
'''
# 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
    createEntry = input("Enter Customer (Yes/No) : ")
    createEntry = createEntry[0].lower()

    if createEntry == "n":

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

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

        # Add the dictionary to the array
        customers.append({'fName': fName, 'lName': lName})

# Print out customer list
for cust in customers:
    print(cust['fName'], cust['lName'])

# ----- RECURSIVE FUNCTIONS -----
# A function that refers to itself is a recursive function

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

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))

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

# ----- PROBLEM : 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

'''
Sample Run Though 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
```

```
'''  
  
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))
```

```
1  # ----- READING & WRITING TEXT -----
2
3  # The os module provides methods for file processing
4  import os
5
6  # We are able to store data for later use in files
7
8  # You can create or use an already created file with open
9
10 # If you use w (write) for mode then the file is
11 # overwritten.
12 # If you use a (append) you add to the end of the file
13
14 # Text is stored using unicode where numbers represent
15 # all possible characters
16
17 # We start the code with with which guarantees the file
18 # will be closed if the program crashes
19 with open("mydata.txt", mode="w", encoding="utf-8") as myFile:
20
21     # You can write to the file with write
22     # It doesn't add a newline
23     myFile.write("Some random text\nMore random text\nAnd some more")
24
25
26 # Open the file for reading
27 # You don't have to provide a mode because it is
28 # read by default
29 with open("mydata.txt", encoding="utf-8") as myFile:
30
31     # We can read data in a few ways
32     # 1. read() reads everything into 1 string
33     # 2. readline() reads everything including the first newline
34     # 3. readlines() returns a list of every line which includes
35     # each newline
36
37     # Use read() to get everything at once
38     print(myFile.read())
39
40 # Find out if the file is closed
41 print(myFile.closed)
42
43 # Get the file name
44 print(myFile.name)
45
46 # Get the access mode of the file
47 print(myFile.mode)
48
49 # Rename our file
50 os.rename("mydata.txt", "mydata2.txt")
51
52 # Delete a file
53 # os.remove("mydata.dat")
54
55 # Create a directory
56 # os.mkdir("mydir")
57
58 # Change directories
59 # os.chdir("mydir")
60
61 # Display current directory
62 print("Current Directory :", os.getcwd())
63
64 # Remove a directory, but 1st move back 1 directory
65 # os.chdir("..")
66 # os.rmdir("mydir")
67
68 # ----- PROBLEM : Fibonacci sequence -----
69 # Previously we generated 1 number in the
```

```
70 # Fibonacci sequence. This time ask the user to define
71 # how many numbers they want and display them
72 # The formula for calculating the Fibonacci sequence is
73 #  $F_n = F_{n-1} + F_{n-2}$ 
74 # Where  $F_0 = 0$  and  $F_1 = 1$ 
75
76 # Sample Output
77 '''
78 How many Fibonacci values should be found : 30
79 1
80 1
81 2
82 3
83 5
84 All Done
85 '''
86
87 def fib(num):
88     if num == 0:
89         return 0
90     elif num == 1:
91         return 1
92     else:
93         result = fib(num - 1) + fib(num - 2)
94         return result
95
96 numFibValues = int(input("How many Fibonacci values should be found : "))
97
98 i = 1
99
100 # While i is less then the number of values requested
101 # continue to find more
102 while i < numFibValues:
103
104     # Call the fib()
105     fibValue = fib(i)
106
107     print(fibValue)
108
109     i += 1
110
111 print("All Done")
112
113
114 # ----- READ ONE LINE AT A TIME -----
115 # You can read 1 line at a time with readline()
116
117 # Open the file
118 with open("mydata2.txt", encoding="utf-8") as myFile:
119
120     lineNum = 1
121
122     # We'll use a while loop that loops until the data
123     # read is empty
124     while True:
125         line = myFile.readline()
126
127         # line is empty so exit
128         if not line:
129             break
130
131         print("Line", lineNum, " :", line, end="")
132
133         lineNum += 1
134
135 # ----- PROBLEM : ANALYZE THE FILE -----
136 # As you cycle through each line output the number of
137 # words and average word length
138 '''
```

```
139 Line 1
140 Number of Words : 3
141 Avg Word Length : 4.7
142 Line 2
143 Number of Words : 3
144 Avg Word Length : 4.7
145 '''
146
147 with open("mydata2.txt", encoding="utf-8") as myFile:
148
149     lineNum = 1
150
151     while True:
152         line = myFile.readline()
153
154         # line is empty so exit
155         if not line:
156             break
157
158         print("Line", lineNum)
159
160         # Put the words in a list using the space as
161         # the boundary between words
162         wordList = line.split()
163
164         # Get the number of words with len()
165         print("Number of Words :", len(wordList))
166
167         # Incremented for each character
168         charCount = 0
169
170         for word in wordList:
171             for char in word:
172                 charCount += 1
173
174         # Divide to find the answer
175         avgNumChars = charCount/len(wordList)
176
177         # Use format to limit to 2 decimals
178         print("Avg Word Length : {:.2}".format(avgNumChars))
179
180         lineNum += 1
181
182 # ----- TUPLES -----
183 # A Tuple is like a list, but their values can't be changed
184 # Tuples are surrounded with parentheses instead of
185 # square brackets
186
187 myTuple = (1, 2, 3, 5, 8)
188
189 # Get a value with an index
190 print("1st Value :", myTuple[0])
191
192 # Get a slice from the 1st index up to but not including
193 # the 3rd
194 print(myTuple[0:3])
195
196 # Get the number of items in a Tuple
197 print("Tuple Length :", len(myTuple))
198
199 # Join or concatenate tuples
200 moreFibs = myTuple + (13, 21, 34)
201
202 # Check if a value is in a Tuple
203 print("34 in Tuple :", 34 in moreFibs)
204
205 # Iterate through a tuple
206 for i in moreFibs:
207     print(i)
```

```
208
209 # Convert a List into a Tuple
210 aList = [55, 89, 144]
211 aTuple = tuple(aList)
212
213 # Convert a Tuple into a List
214 aList = list(aTuple)
215
216 # Get max and minimum value
217 print("Min :", min(aTuple))
218 print("Max :", max(aTuple))
```

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

# We are able to store data for later use in files

# 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")

# 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")

# ----- PROBLEM : Fibonacci sequence -----
# Previously we generated 1 number in the
# Fibonacci sequence. This time ask the user to define
# how many numbers they want and display them
# The formula for calculating the Fibonacci sequence is
#  $F_n = F_{n-1} + F_{n-2}$ 
# Where  $F_0 = 0$  and  $F_1 = 1$ 
```

```

# Sample Output
'''
How many Fibonacci values should be found : 30
1
1
2
3
5
All Done
'''

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")

# ----- READ ONE LINE AT A TIME -----
# You can read 1 line at a time with readline()

# 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

# ----- PROBLEM : ANALYZE THE FILE -----
# As you cycle through each line output the number of
# words and average word length
'''
Line 1
Number of Words : 3
Avg Word Length : 4.7
Line 2
Number of Words : 3
Avg Word Length : 4.7
'''

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 -----
# A Tuple is like a list, but their values can't be changed
# Tuples are surrounded with parentheses instead of
# square brackets

myTuple = (1, 2, 3, 5, 8)

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

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

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

# Join or concatenate tuples
moreFibs = myTuple + (13, 21, 34)

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

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

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

# Convert a Tuple into a List
aList = list(aTuple)

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

If you like videos like this consider donating on [Patreon](#).

Code & Transcript

```
Learn to Program 9
2
3 # Real world objects have attributes and capabilities
4
5 # A dog for example has the attributes of height, weight
6 # favorite food, etc.
7
8 # It has the capability to run, bark, scratch, etc.
9
10 # In object oriented programming we model real world objects
11 # by defining the attributes (fields) and capabilities (methods)
12 # that they have.
13
14 # A class is the template used to model these objects
15 # Here we will model a Dog object
16
17 class Dog:
18
19     # The init method is called to create an object
20     # We give default values for the fields if none
21     # are provided
22     def __init__(self, name="", height=0, weight=0):
23
24         # self allows an object to refer to itself
25         # It is like how you refer to yourself with my
26
27         # We will take the values passed in and assign
28         # them to the new Dog objects fields (attributes)
29         self.name = name
30         self.height = height
31         self.weight = weight
32
33         # Define what happens when the Dog is asked to
34         # demonstrate its capabilities
35
36     def run(self):
37         print("{} the dog runs".format(self.name))
38
39     def eat(self):
40         print("{} the dog eats".format(self.name))
41
42     def bark(self):
43         print("{} the dog barks".format(self.name))
44
45
46 def main():
47
48     # Create a new Dog object
49     spot = Dog("Spot", 66, 26)
50
51     spot.bark()
52
53     bowser = Dog()
54
55 main()
56
57
58 # ----- GETTERS & SETTERS -----
59 # Getters and Setters are used to protect our objects
60 # from assigning bad fields or for providing improved
61 # output
62
63 class Square:
```

```
64     def __init__(self, height="0", width="0"):  
65         self.height = height  
66         self.width = width  
67  
68         # This is the getter  
69         @property  
70         def height(self):  
71             print("Retrieving the height")  
72  
73             # Put a __ before this private field  
74             return self.__height  
75  
76         # This is the setter  
77         @height.setter  
78         def height(self, value):  
79  
80             # We protect the height from receiving a bad value  
81             if value.isdigit():  
82  
83                 # Put a __ before this private field  
84                 self.__height = value  
85             else:  
86                 print("Please only enter numbers for height")  
87  
88         # This is the getter  
89         @property  
90         def width(self):  
91             print("Retrieving the width")  
92             return self.__width  
93  
94         # This is the setter  
95         @width.setter  
96         def width(self, value):  
97             if value.isdigit():  
98                 self.__width = value  
99             else:  
100                 print("Please only enter numbers for width")  
101  
102         def getArea(self):  
103             return int(self.__width) * int(self.__height)  
104  
105  
106 def main():  
107     aSquare = Square()  
108  
109     height = input("Enter height : ")  
110     width = input("Enter width : ")  
111  
112     aSquare.height = height  
113     aSquare.width = width  
114  
115     print("Height :", aSquare.height)  
116     print("Width :", aSquare.width)  
117  
118     print("The Area is :", aSquare.getArea())  
119  
120  
121 main()  
122  
123 # ----- WARRIORS BATTLE -----  
124 # We will create a game with this sample output  
125 '''  
126 Sam attacks Paul and deals 9 damage  
127 Paul is down to 10 health  
128 Paul attacks Sam and deals 7 damage  
129 Sam is down to 7 health  
130 Sam attacks Paul and deals 19 damage  
131 Paul is down to -9 health  
132 Paul has Died and Sam is Victorious
```

```
133 Game Over
134 '''
135
136 # We will create a Warrior & Battle class
137
138 import random
139 import math
140
141 # Warriors will have names, health, and attack and block maximums
142 # They will have the capabilities to attack and block random amounts
143 class Warrior:
144     def __init__(self, name="warrior", health=0, attkMax=0, blockMax=0):
145         self.name = name
146         self.health = health
147         self.atkMax = attkMax
148         self.blockMax = blockMax
149
150     def attack(self):
151         # Randomly calculate the attack amount
152         # random() returns a value from 0.0 to 1.0
153         attkAmt = self.atkMax * (random.random() + .5)
154
155         return attkAmt
156
157     def block(self):
158
159         # Randomly calculate how much of the attack was blocked
160         blockAmt = self.blockMax * (random.random() + .5)
161
162         return blockAmt
163
164 # The Battle class will have the capability to loop until 1 Warrior dies
165 # The Warriors will each get a turn to attack each turn
166
167 class Battle:
168
169     def startFight(self, warrior1, warrior2):
170
171         # Continue looping until a Warrior dies switching back and
172         # forth as the Warriors attack each other
173         while True:
174             if self.getAttackResult(warrior1, warrior2) == "Game Over":
175                 print("Game Over")
176                 break
177
178             if self.getAttackResult(warrior2, warrior1) == "Game Over":
179                 print("Game Over")
180                 break
181
182         # A function will receive each Warrior that will attack the other
183         # Have the attack and block amounts be integers to make the results clear
184         # Output the results of the fight as it goes
185         # If a Warrior dies return that result to end the looping in the
186         # above function
187
188         # Make this method static because we don't need to use self
189         @staticmethod
190         def getAttackResult(warriorA, warriorB):
191             warriorAAtkAmt = warriorA.attack()
192
193             warriorBBlockAmt = warriorB.block()
194
195             damage2WarriorB = math.ceil(warriorAAtkAmt - warriorBBlockAmt)
196
197             warriorB.health = warriorB.health - damage2WarriorB
198
199             print("{} attacks {} and deals {} damage".format(warriorA.name,
200                                                                warriorB.name, dama
201
```

```
202         print("{} is down to {} health".format(warriorB.name,
203                                                  warriorB.health))
204
205     if warriorB.health <= 0:
206         print("{} has Died and {} is Victorious".format(warriorB.name,
207                                                         warriorA.name))
208
209         return "Game Over"
210     else:
211         return "Fight Again"
212
213
214 def main():
215
216     # Create 2 Warriors
217     paul = Warrior("Paul", 50, 20, 10)
218     sam = Warrior("Sam", 50, 20, 10)
219
220     # Create Battle object
221     battle = Battle()
222
223     # Initiate Battle
224     battle.startFight(paul, sam)
225
226 main()
```

```
1  # When we create a class we can inherit all of the fields and methods
2  # from another class. This is called inheritance.
3
4  # The class that inherits is called the subclass and the class we
5  # inherit from is the super class
6
7  # This will be our super class
8  class Animal:
9
10     def __init__(self, birthType="Unknown",
11                  appearance="Unknown",
12                  blooded="Unknown"):
13         self.__birthType = birthType
14         self.__appearance = appearance
15         self.__blooded = blooded
16
17     # The getter method
18     @property
19     def birthType(self):
20
21         # When using getters and setters don't forget the __
22         return self.__birthType
23
24     @birthType.setter
25     def birthType(self, birthType):
26         self.__birthType = birthType
27
28     @property
29     def appearance(self):
30         return self.__appearance
31
32     @appearance.setter
33     def appearance(self, appearance):
34         self.__appearance = appearance
35
36     @property
37     def blooded(self):
38         return self.__blooded
39
40     @blooded.setter
41     def blooded(self, blooded):
42         self.__blooded = blooded
43
44     # Can be used to cast our object as a string
45     # type(self).__name__ returns the class name
46     def __str__(self):
47         return "A {} is {} it is {} it is " \
48                "{}".format(type(self).__name__,
49                             self.birthType,
50                             self.appearance,
51                             self.blooded)
52
53     # Create a Mammal class that inherits from Animal
54     # You can inherit from multiple classes by separating
55     # the classes with a comma in the parentheses
56     class Mammal(Animal):
57         def __init__(self, birthType="born alive",
58                      appearance="hair or fur",
59                      blooded="warm blooded",
60                      nurseYoung=True):
61
62             # Call for the super class to initialize fields
63             Animal.__init__(self, birthType,
64                             appearance,
65                             blooded)
66
67             self.__nurseYoung = nurseYoung
68
69     # We can extend the subclasses
```

```
70 @property
71 def nurseYoung(self):
72     return self.__nurseYoung
73
74 @nurseYoung.setter
75 def appearance(self, nurseYoung):
76     self.__nurseYoung = nurseYoung
77
78 # Overwrite __str__
79 # You can use super() to refer to the superclass
80 def __str__(self):
81     return super().__str__() + " and it is {} they nurse " \
82         "their young".format(self.nurseYoung)
83
84
85 class Reptile(Animal):
86     def __init__(self, birthType="born in an egg or born alive",
87                 appearance="dry scales",
88                 blooded="cold blooded"):
89
90         # Call for the super class to initialize fields
91         Animal.__init__(self, birthType,
92                         appearance,
93                         blooded)
94
95         # Operator overloading isn't necessary in Python because
96         # Python allows you to enter unknown numbers of values
97         # Always make sure self is the first attribute in your
98         # class methods
99     def sumAll(self, *args):
100         sum = 0
101
102         for i in args:
103
104             sum += i
105
106         return sum
107
108
109 def main():
110     animal1 = Animal("born alive")
111
112     print(animal1.birthType)
113
114     # Call __str__()
115     print(animal1)
116     print()
117
118     mammal1 = Mammal()
119
120     print(mammal1)
121
122     print(mammal1.birthType)
123     print(mammal1.appearance)
124     print(mammal1.blooded)
125     print(mammal1.nurseYoung)
126     print()
127
128     reptile1 = Reptile()
129
130     print(reptile1.birthType)
131     print(reptile1.appearance)
132     print(reptile1.blooded)
133     print()
134
135     # Operator overloading in Python
136     print("Sum : {}".format(reptile1.sumAll(1,2,3,4,5)))
137
138     # Polymorphism in Python works differently from other
```

```

139 # languages in that functions accept any object
140 # and expect that object to provide the needed method
141
142 # This isn't something to dwell on. Just know that
143 # if you call on a method for an object that the
144 # method just needs to exist for that object to work.
145 # Polymorphism is a big deal in other languages that
146 # are statically typed (type is defined at declaration)
147 # but because Python is dynamically typed (type defined
148 # when a value is assigned) it doesn't matter as much.
149
150 def getBirthType(theObject):
151     print("The {} is {}".format(type(theObject).__name__,
152                                 theObject.birthType))
153
154 getBirthType(mammal1)
155 getBirthType(reptile1)
156
157
158
159 main()
160
161 # ----- MAGIC METHODS -----
162 # Magic methods are surrounded by double underscores
163 # We can use magic methods to define how operators
164 # like +, -, *, /, ==, >, <, etc. will work with our
165 # custom objects
166
167 # Magic methods are used for operator overloading
168 # in Python
169
170 # __eq__ : Equal
171 # __ne__ : Not Equal
172 # __lt__ : Less Than
173 # __gt__ : Greater Than
174 # __le__ : Less Than or Equal
175 # __ge__ : Greater Than or Equal
176 # __add__ : Addition
177 # __sub__ : Subtraction
178 # __mul__ : Multiplication
179 # __div__ : Division
180 # __mod__ : Modulus
181
182 class Time:
183     def __init__(self, hour=0, minute=0, second=0):
184         self.hour = hour
185         self.minute = minute
186         self.second = second
187
188     # Magic method that defines the string format of the object
189     def __str__(self):
190
191         # :02d adds a leading zero to have a minimum of 2 digits
192         return "{}:{:02d}:{:02d}".format(self.hour, self.minute, self.second)
193
194     def __add__(self, other_time):
195
196         new_time = Time()
197
198         # ----- PROBLEM -----
199         # How would you go about adding 2 times together?
200
201         # Add the seconds and correct if sum is >= 60
202         if (self.second + other_time.second) >= 60:
203             self.minute += 1
204             new_time.second = (self.second + other_time.second) - 60
205         else:
206             new_time.second = self.second + other_time.second
207

```



```
208     # Add the minutes and correct if sum is >= 60
209     if (self.minute + other_time.minute) >= 60:
210         self.hour += 1
211         new_time.minute = (self.minute + other_time.minute) - 60
212     else:
213         new_time.minute = self.minute + other_time.minute
214
215     # Add the minutes and correct if sum is > 60
216
217     if (self.hour + other_time.hour) > 24:
218         new_time.hour = (self.hour + other_time.hour) - 24
219     else:
220         new_time.hour = self.hour + other_time.hour
221
222     return new_time
223
224
225 def main():
226     time1 = Time(1, 20, 30)
227
228     print(time1)
229
230     time2 = Time(24, 41, 30)
231
232     print(time1 + time2)
233
234     # For homework get the Time objects to work for the other
235     # mathematical and comparison operators
236
237
238
239 main()
```

e this consider contributing on Patreon.

Code & Transcript

```
Learn to Program 11 Python
2
3 # Static methods allow access without the need to initialize
4 # a class. They should be used as utility methods, or when
5 # a method is needed, but it doesn't make sense for the real
6 # world object to be able to perform a task
7
8 class Sum:
9
10     # You use the static method decorator to define that a
11     # method is static
12     @staticmethod
13     def getSum(*args):
14
15         sum = 0
16
17         for i in args:
18             sum += i
19
20         return sum
21
22 def main():
23
24     # Call a static method by proceeding it with its class
25     # name
26     print("Sum :", Sum.getSum(1,2,3,4,5))
27
28
29 main()
30
31
32 # ----- STATIC VARIABLES -----
33 # Fields declared in a class, but outside of any method
34 # are static variables. Their value is shared by every
35 # object of that class
36
37 class Dog:
38
39     # This is a static variable
40     num_of_dogs = 0
41
42     def __init__(self, name="Unknown"):
43         self.name = name
44
45         # You reference the static variable by proceeding
46         # it with the class name
47         Dog.num_of_dogs += 1
48
49     @staticmethod
50     def getNumOfDogs():
51         print("There are currently {} dogs".format(Dog.num_of_dogs))
52
53 def main():
54
55     spot = Dog("Spot")
56
57     doug = Dog("Doug")
58
59     spot.getNumOfDogs()
60
61
62 main()
63
```

```
64 # ----- MODULES -----
65 # Your Python programs will contain a main program that
66 # includes your main function. Then you will create many
67 # modules in separate files. Modules also end with .py
68 # just like any other Python file
69
70 # ----- sum.py -----
71 def getSum(*args):
72     sum = 0
73
74     for i in args:
75         sum += i
76
77     return sum
78
79 # ----- End of sum.py -----
80
81 # You can import by listing the file name minus the py
82 import sum
83
84 # Get access to functions by proceeding with the file
85 # name and then the function you want
86 print("Sum :", sum.getSum(1,2,3,4,5))
87
88 # ----- FROM -----
89
90 # You can use from to copy specific functions from a module
91 # You can use from sum import * to import all functions
92 # You can import multiple functions by listing them after
93 # import separated by commas
94 from sum import getSum
95
96 # You don't have to reference the module name now
97 print("Sum :", getSum(1,2,3,4,5))
98
99
100 # ----- EXCEPTION HANDLING -----
101 # Exceptions are triggered either when an error occurs
102 # or when you want them to.
103
104 # We use exceptions are used to handle errors, execute
105 # specific code when code generates something out of
106 # the ordinary, to always execute code when something
107 # happens (close a file that was opened),
108
109 # When an error occurs you stop executing code and jump
110 # to execute other code that responds to that error
111
112 # Let's handle an IndexError exception that is
113 # triggered when you try to access an index in a list
114 # that doesn't exist
115
116 # Surround a potential exception with try
117 try:
118     aList = [1,2,3]
119
120     print(aList[3])
121
122 # Catch the exception with except followed by the
123 # exception you want to catch
124
125 # You can catch multiple exceptions by separating them
126 # with commas inside parentheses
127 # except (IndexError, NameError):
128 except IndexError:
129     print("Sorry that index doesn't exist")
130
131 # If the exception wasn't caught above this will
132 # catch all others
```

```
133 except:
134     print("An unknown error occurred")
135
136 # ----- CUSTOM EXCEPTIONS -----
137
138 # Lets trigger an exception if the user enters a
139 # name that contains a number
140
141 # Although you won't commonly create your own exceptions
142 # this is how you do it
143
144 # Create a class that inherits from Exception
145 class DogNameError(Exception):
146
147     def __init__(self, *args, **kwargs):
148         Exception.__init__(self, *args, **kwargs)
149
150 try:
151     dogName = input("What is your dogs name : ")
152
153     if any(char.isdigit() for char in dogName):
154
155         # Raise your own exception
156         # You can raise the built in exceptions as well
157         raise DogNameError
158
159 except DogNameError:
160     print("Your dogs name can't contain a number")
161
162 # ----- FINALLY & ELSE -----
163 # finally is used when you always want certain code to
164 # execute whether an exception is raised or not
165
166 num1, num2 = input("Enter to values to divide : ").split()
167
168 try:
169     quotient = int(num1) / int(num2)
170     print("{} / {} = {}".format(num1, num2, quotient))
171
172 except ZeroDivisionError:
173     print("You can't divide by zero")
174
175 # else is only executed if no exception was raised
176 else:
177     print("You didn't raise an exception")
178
179 finally:
180     print("I execute no matter what")
181
182 # ----- PROBLEM EXCEPTIONS & FILES -----
183 # 1. Create a file named mydata2.txt and put data in it
184 # 2. Using what you learned in part 8 and Google to find
185 # out how to open a file without with try to open the
186 # file in a try block
187 # 3. Catch the FileNotFoundError exception
188 # 4. In else print the file contents
189 # 5. In finally close the file
190 # 6. Try to open the nonexistent file mydata3.txt and
191 # test to see if you caught the exception
192
193 try:
194     myFile = open("mydata2.txt", encoding="utf-8")
195
196 # We can use as to access data and methods in the
197 # exception class
198 except FileNotFoundError as ex:
199     print("That file was not found")
200
201     # Print out further data on the exception
```

```
202     print(ex.args)
203
204 else:
205     print("File :", myFile.read())
206     myFile.close()
207
208 finally:
209     print("Finished Working with File")
```

```
# ----- FUNCTIONS AS OBJECTS -----

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 getFuncMultByNum()
    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

listOfFuncs = [times_two, generated_func]

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

# ----- PROBLEM -----
# Create a function that receives a list and a function
# The function passed will return True or False if a list
# value is odd.
# The surrounding function will return a list of odd
# numbers

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

def change_list(list, func):
    oddList = []

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

    return oddList

aList = range(1, 21)

print(change_list(aList, 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__)

# ----- ANONYMOUS FUNCTIONS : LAMBDA -----
# 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

# Add values
sum = lambda x, y : x + y

print("Sum :", sum(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
powerList = [lambda x: x ** 2,
             lambda x: x ** 3,
             lambda x: x ** 4]

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

# You can also store lambdas in dictionaries

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
attackKey = random.choice(list(attack.keys()))

attack[attackKey]()

# ----- PROBLEM -----
# 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

# Create the list
flipList = []

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

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

# ----- MAP -----
# Map allows us to execute a function on each item in a list

# Generate a list from 1 to 10
oneTo10 = 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, oneTo10)))

# You could do the same thing with a lambda
print(list(map((lambda x: x * 3), oneTo10)))

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

# ----- FILTER -----
# Filter selects items from a list based on a function

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

# ----- PROBLEM -----
# Find the multiples of 9 from a random 100 value list with
# values between 1 and 1000

# Generate a random list with randint between 1 and 1000
# Use range to generate 100 values
randList = 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), randList)))

# ----- REDUCE -----
# Reduce receives a list and returns a single result

# 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)))
```


If you value videos like this consider [donating a \\$1 on Patreon](#).

Cheat Sheet & Transcript

```

Learn to Program 14
2  # multiple programs at once.
3
4  # Threads actually take turns executing. While one
5  # executes the other sleeps until it is its turn
6  # to execute.
7
8  import threading
9  import time
10 import random
11
12 def executeThread(i):
13
14     # strftime or string formatted time allows you to
15     # define how the time is displayed.
16     # You could include the date with
17     # strftime("%Y-%m-%d %H:%M:%S", gmtime())
18
19     # Print when the thread went to sleep
20     print("Thread {} sleeps at {}".format(i,
21         time.strftime("%H:%M:%S", time.gmtime()))
22
23     # Generate a random sleep period of between 1 and
24     # 5 seconds
25     randSleepTime = random.randint(1, 5)
26
27     # Pauses execution of code in this function for
28     # a few seconds
29     time.sleep(randSleepTime)
30
31     # Print out info after the sleep time
32     print("Thread {} stops sleeping at {}".format(i,
33         time.strftime("%H:%M:%S", time.gmtime()))
34
35 for i in range(10):
36
37     # Each time through the loop a Thread object is created
38     # You pass it the function to execute and any
39     # arguments to pass to that method
40     # The arguments passed must be a sequence which
41     # is why we need the comma with 1 argument
42     thread = threading.Thread(target=executeThread, args=(i,))
43     thread.start()
44
45     # Display active threads
46     # The extra 1 is this for loop executing in the main
47     # thread
48     print("Active Threads :", threading.activeCount())
49
50     # Returns a list of all active thread objects
51     print("Thread Objects :", threading.enumerate())
52
53
54 # ----- SUBCLASSING THREAD -----
55 # You can subclass the Thread object and then define
56 # what happens each time a new thread is executed
57 # or run
58
59 class CustThread(threading.Thread):
60
61     def __init__(self, name):
62         threading.Thread.__init__(self)
63

```

```
64         self.name = name
65
66     def run(self):
67
68         getTime(self.name)
69
70         print("Thread", self.name, "Execution Ends")
71
72 def getTime(name):
73     print("Thread {} sleeps at {}".format(name,
74         time.strftime("%H:%M:%S", time.gmtime()))))
75
76     randSleepTime = random.randint(1, 5)
77
78     time.sleep(randSleepTime)
79
80 # Create thread objects
81 thread1 = CustThread("1")
82 thread2 = CustThread("2")
83
84 # Start thread execution of run()
85 thread1.start()
86 thread2.start()
87
88 # Check if thread is alive
89 print("Thread 1 Alive :", thread1.is_alive())
90 print("Thread 2 Alive :", thread2.is_alive())
91
92 # Get thread name
93 # You can change it with setName()
94 print("Thread 1 Name :", thread1.getName())
95 print("Thread 2 Name :", thread2.getName())
96
97 # Wait for threads to exit
98 thread1.join()
99 thread2.join()
100
101 print("Execution Ends")
102
103
104 # ----- SYNCHRONIZING THREADS -----
105 # You can lock other threads from executing
106
107 # If we try to model a bank account we have to make sure
108 # the account is locked down during a transaction so
109 # that if more then 1 person tries to withdrawal money at
110 # once we don't give out more money then is in the account
111
112 class BankAccount (threading.Thread):
113
114     acctBalance = 100
115
116     def __init__(self, name, moneyRequest):
117         threading.Thread.__init__(self)
118         self.name = name
119         self.moneyRequest = moneyRequest
120
121     def run(self):
122         # Get lock to keep other threads from accessing the account
123         threadLock.acquire()
124
125         # Call the static method
126         BankAccount.getMoney(self)
127
128         # Release lock so other thread can access the account
129         threadLock.release()
130
131     @staticmethod
132     def getMoney(customer):
```

```
133         print("{} tries to withdrawal ${} at {}".format(customer.name,
134               customer.moneyRequest,
135               time.strftime("%H:%M:%S", time.gmtime()))
136
137         if BankAccount.acctBalance - customer.moneyRequest > 0:
138             BankAccount.acctBalance -= customer.moneyRequest
139             print("New account balance is : {}".format(BankAccount.acctBalance))
140         else:
141             print("Not enough money in the account")
142             print("Current balance : {}".format(BankAccount.acctBalance))
143
144         time.sleep(3)
145
146     # Create a lock to be used by threads
147     threadLock = threading.Lock()
148
149     # Create new threads
150     doug = BankAccount("Doug", 1)
151     paul = BankAccount("Paul", 100)
152     sally = BankAccount("Sally", 50)
153
154     # Start new Threads
155     doug.start()
156     paul.start()
157     sally.start()
158
159     # Have threads wait for previous threads to terminate
160     doug.join()
161     paul.join()
162     sally.join()
163
164     print("Execution Ends")
```

```
to Program 15
2 # strings in very powerful ways.
3 # They work in almost exactly the same way in every
4 # programming language as well.
5
6 # Regular Expressions (Regex) are used to
7 # 1. Search for a specific string in a large amount of data
8 # 2. Verify that a string has the proper format (Email, Phone #)
9 # 3. Find a string and replace it with another string
10 # 4. Format data into the proper form for importing for example
11
12 # import the Regex module
13 import re
14
15 # ----- Was a Match Found -----
16
17 # Search for ape in the string
18 if re.search("ape", "The ape was at the apex"):
19     print("There is an ape")
20
21 # ----- Get All Matches -----
22
23 # findall() returns a list of matches
24 # . is used to match any 1 character or space
25 allApes = re.findall("ape.", "The ape was at the apex")
26
27 for i in allApes:
28     print(i)
29
30 # finditer returns an iterator of matching objects
31 # You can use span to get the location
32
33 theStr = "The ape was at the apex"
34
35 for i in re.finditer("ape.", theStr):
36
37     # Span returns a tuple
38     locTuple = i.span()
39
40     print(locTuple)
41
42     # Slice the match out using the tuple values
43     print(theStr[locTuple[0]:locTuple[1]])
44
45 # ----- Match 1 of Several Letters -----
46
47 # Square brackets will match any one of the characters between
48 # the brackets not including upper and lowercase varieties
49 # unless they are listed
50
51 animalStr = "Cat rat mat fat pat"
52
53 allAnimals = re.findall("[crmf]at", animalStr)
54 for i in allAnimals:
55     print(i)
56
57 print()
58
59 # We can also allow for characters in a range
60 # Remember to include upper and lowercase letters
61 someAnimals = re.findall("[c-mC-M]at", animalStr)
62 for i in someAnimals:
63     print(i)
64
65 print()
66
67 # Use ^ to denote any character but whatever characters are
68 # between the brackets
69 someAnimals = re.findall("[^Cr]at", animalStr)
```

```
70 for i in someAnimals:
71     print(i)
72
73 print()
74
75 # ----- Replace All Matches -----
76
77 # Replace matching items in a string
78
79 owlFood = "rat cat mat pat"
80
81 # You can compile a regex into pattern objects which
82 # provide additional methods
83 regex = re.compile("[cr]at")
84
85 # sub() replaces items that match the regex in the string
86 # with the 1st attribute string passed to sub
87 owlFood = regex.sub("owl", owlFood)
88
89 print(owlFood)
90
91 # ----- Solving Backslash Problems -----
92
93 # Regex use the backslash to designate special characters
94 # and Python does the same inside strings which causes
95 # issues.
96
97 # Let's try to get "\\stuff" out of a string
98
99 randStr = "Here is \\stuff"
100
101 # This won't find it
102 print("Find \\stuff : ", re.search("\\stuff", randStr))
103
104 # This does, but we have to put in 4 slashes which is
105 # messy
106 print("Find \\stuff : ", re.search("\\\\stuff", randStr))
107
108 # You can get around this by using raw strings which
109 # don't treat backslashes as special
110 print("Find \\stuff : ", re.search(r"\\stuff", randStr))
111
112 # ----- Matching Any Character -----
113 # We saw that . matches any character, but what if we
114 # want to match a period. Backslash the period
115 # You do the same with [, ] and others
116
117 randStr = "F.B.I. I.R.S. CIA"
118
119 print("Matches :", len(re.findall(".\\.\\.\\.\\.", randStr)))
120
121 # ----- Matching Whitespace -----
122 # We can match many whitespace characters
123
124 randStr = """This is a long
125 string that goes
126 on for many lines"""
127
128 print(randStr)
129
130 # Remove newlines
131 regex = re.compile("\n")
132
133 randStr = regex.sub(" ", randStr)
134
135 print(randStr)
136
137 # You can also match
138 # \b : Backspace
```

```
139 # \f : Form Feed
140 # \r : Carriage Return
141 # \t : Tab
142 # \v : Vertical Tab
143
144 # You may need to remove \r\n on Windows
145
146 # ----- Matching Any Single Number -----
147 # \d can be used instead of [0-9]
148 # \D is the same as [^0-9]
149
150 randStr = "12345"
151
152 print("Matches :", len(re.findall("\d", randStr)))
153
154 # ----- Matching Multiple Numbers -----
155 # You can match multiple digits by following the \d with {numOfValues}
156
157 # Match 5 numbers only
158 if re.search("\d{5}", "12345"):
159     print("It is a zip code")
160
161 # You can also match within a range
162 # Match values that are between 5 and 7 digits
163 numStr = "123 12345 123456 1234567"
164
165 print("Matches :", len(re.findall("\d{5,7}", numStr)))
166
167 # ----- Matching Any Single Letter or Number -----
168 # \w is the same as [a-zA-Z0-9_]
169 # \W is the same as [^a-zA-Z0-9_]
170
171 phNum = "412-555-1212"
172
173 # Check if it is a phone number
174 if re.search("\w{3}-\w{3}-\w{4}", phNum):
175     print("It is a phone number")
176
177 # Check for valid first name between 2 and 20 characters
178 if re.search("\w{2,20}", "Ultraman"):
179     print("It is a valid name")
180
181 # ----- Matching WhiteSpace -----
182 # \s is the same as [\f\n\r\t\v]
183 # \S is the same as [^\f\n\r\t\v]
184
185 # Check for valid first and last name with a space
186 if re.search("\w{2,20}\s\w{2,20}", "Toshio Muramatsu"):
187     print("It is a valid full name")
188
189 # ----- Matching One or More -----
190 # + matches 1 or more characters
191
192 # Match a followed by 1 or more characters
193 print("Matches :", len(re.findall("a+", "a as ape bug")))
194
195 # ----- Problem -----
196 # Create a Regex that matches email addresses from a list
197 # 1. 1 to 20 lowercase and uppercase letters, numbers, plus ._%+-
198 # 2. An @ symbol
199 # 3. 2 to 20 lowercase and uppercase letters, numbers, plus .-
200 # 4. A period
201 # 5. 2 to 3 lowercase and uppercase letters
202
203 emailList = "db@aol.com m@.com @apple.com db@.com"
204
205 print("Email Matches :", len(re.findall("[\w._%+-]{1,20}@[ \w.-]{2,20}.[A-Za-
206                                     emailList)))
```

If you like videos like this consider donating a \$1 on [Patreon](#).

Code & Transcript

```
Learn to Program 16 Python
2
3 # Did you find a match
4 # if re.search("REGEX", yourString)
5
6 # Get list of matches
7 # print("Matches :", len(re.findall("REGEX", yourString)))
8
9 # Get a pattern object
10 # regex = re.compile("REGEX")
11
12 # Substitute the match
13 # yourString = regex.sub("substitution", yourString)
14
15 # [ ] : Match what is in the brackets
16 # [^ ] : Match anything not in the brackets
17 # . : Match any 1 character or space
18 # + : Match 1 or more of what proceeds
19 # \n : Newline
20 # \d : Any 1 number
21 # \D : Anything but a number
22 # \w : Same as [a-zA-Z0-9_]
23 # \W : Same as [^a-zA-Z0-9_]
24 # \s : Same as [\f\n\r\t\v]
25 # \S : Same as [^\f\n\r\t\v]
26 # {5} : Match 5 of what proceeds the curly brackets
27 # {5,7} : Match values that are between 5 and 7 in length
28
29 # ----- Matching Zero or One -----
30
31 randStr = "cat cats"
32
33 regex = re.compile("[cat]+s?")
34
35 matches = re.findall(regex, randStr)
36
37 # Match cat or cats
38 print("Matches :", len(matches))
39
40 for i in matches:
41     print(i)
42
43 # ----- Matching Zero or More -----
44 # * matches zero or more of what proceeds it
45
46 randStr = "doctor doctors doctor's"
47
48 # Match doctor doctors or doctor's
49 regex = re.compile("[doctor]+['s]*")
50
51 matches = re.findall(regex, randStr)
52
53 print("Matches :", len(matches))
54
55 # You can do the same by setting an interval match
56 regex = re.compile("[doctor]+['s']{0,2}")
57
58 matches = re.findall(regex, randStr)
59
60 print("Matches :", len(matches))
61
62 for i in matches:
63     print(i)
```

```
64
65 # ----- PROBLEM -----
66 # On Windows newlines are some times \n and other times \r\n
67 # Create a regex that will grab each of the lines in this
68 # string, print out the number of matches and each line
69
70 longStr = '''Just some words
71 and some more\r
72 and more
73 '''
74
75 print("Matches :", len(re.findall(r"[\w\s]+[\r]?\\n", longStr)))
76
77 matches = re.findall("[\w\s]+[\r]?\\n", longStr)
78
79 for i in matches:
80     print(i)
81
82 # ----- Greedy & Lazy Matching -----
83
84 randStr = "<name>Life On Mars</name><name>Freaks and Geeks</name>"
85
86 # Let's try to grab everything between <name> tags
87 # Because * is greedy (It grabs the biggest match possible)
88 # we can't get what we want, which is each individual tag
89 # match
90 regex = re.compile(r"<name>.*</name>")
91
92 matches = re.findall(regex, randStr)
93
94 print("Matches :", len(matches))
95
96 for i in matches:
97     print(i)
98
99 # We want to grab the smallest match we use *?, +?, or
100 # {n,}? instead
101
102 regex = re.compile(r"<name>.*?</name>")
103
104 matches = re.findall(regex, randStr)
105
106 print("Matches :", len(matches))
107
108 for i in matches:
109     print(i)
110
111 # ----- Word Boundaries -----
112 # We use word boundaries to define where our matches start
113 # and end
114
115 # \b matches the start or end of a word
116
117 # If we want ape it will match ape and the beginning of apex
118 randStr = "ape at the apex"
119
120 regex = re.compile(r"ape")
121
122 # If we use the word boundary
123 regex = re.compile(r"\bape\b")
124
125 matches = re.findall(regex, randStr)
126
127 print("Matches :", len(matches))
128
129 for i in matches:
130     print(i)
131
132 # ----- String Boundaries -----
```



```
133 # ^ : Matches the beginning of a string if outside of
134 #     a [ ]
135 # $ : Matches the end of a string
136
137 # Grab everything from the start of the string to @
138 randStr = "Match everything up to @"
139
140 regex = re.compile(r"^.*[@]")
141
142 matches = re.findall(regex, randStr)
143
144 print("Matches :", len(matches))
145
146 for i in matches:
147     print(i)
148
149 # Grab everything from @ to the end of the line
150 randStr = "@ Get this string"
151
152 regex = re.compile(r"[^@\s].*$")
153
154 matches = re.findall(regex, randStr)
155
156 print("Matches :", len(matches))
157
158 for i in matches:
159     print(i)
160
161 # Grab the 1st word of each line using the the multiline
162 # code which allows for the targeting of each line after
163 # a line break with ^
164 randStr = '''Ape is big
165 Turtle is slow
166 Cheetah is fast'''
167
168 regex = re.compile(r"(?m)^.*?\s")
169
170 matches = re.findall(regex, randStr)
171
172 print("Matches :", len(matches))
173
174 for i in matches:
175     print(i)
176
177 # ----- Subexpressions -----
178 # Subexpressions are parts of a larger expression
179 # If you want to match for a large block, but
180 # only want to return part of it. To do that
181 # surround what you want with ( )
182
183 # Get just the number minus the area code
184 randStr = "My number is 412-555-1212"
185
186 regex = re.compile(r"412-(.*)")
187
188 matches = re.findall(regex, randStr)
189
190 print("Matches :", len(matches))
191
192 for i in matches:
193     print(i)
194
195 # ----- Problem -----
196
197 # Get just the numbers minus the area codes from
198 # this string
199 randStr = "412-555-1212 412-555-1213 412-555-1214"
200
201 regex = re.compile(r"412-(.{8})")
```

```
202
203 matches = re.findall(regex, randStr)
204
205 print("Matches :", len(matches))
206
207 for i in matches:
208     print(i)
209
210 # ----- Multiple Subexpressions -----
211
212 # You can have multiple subexpressions as well
213 # Get both numbers that follow 412 separately
214 randStr = "My number is 412-555-1212"
215
216 regex = re.compile(r"412-(.*)-(.*)")
217
218 matches = re.findall(regex, randStr)
219
220 print("Matches :", len(matches))
221
222 print(matches[0][0])
223 print(matches[0][1])
```

If you think videos like this help consider donating a \$1 on [Patreon](#).

Code & Transcript

```
Learn to Program 17 Python
2
3 # Did you find a match
4 # if re.search("REGEX", yourString)
5
6 # Get list of matches
7 # print("Matches :", len(re.findall("REGEX", yourString)))
8
9 # Get a pattern object
10 # regex = re.compile("REGEX")
11
12 # Substitute the match
13 # yourString = regex.sub("substitution", yourString)
14
15 # [ ] : Match what is in the brackets
16 # [^ ] : Match anything not in the brackets
17 # ( ) : Return surrounded submatch
18 # . : Match any 1 character or space
19 # + : Match 1 or more of what proceeds
20 # ? : Match 0 or 1
21 # * : Match 0 or More
22 # *? : Lazy match the smallest match
23 # \b : Word boundary
24 # ^ : Beginning of String
25 # $ : End of String
26 # \n : Newline
27 # \d : Any 1 number
28 # \D : Anything but a number
29 # \w : Same as [a-zA-Z0-9_]
30 # \W : Same as [^a-zA-Z0-9_]
31 # \s : Same as [\f\n\r\t\v]
32 # \S : Same as [^\f\n\r\t\v]
33 # {5} : Match 5 of what proceeds the curly brackets
34 # {5,7} : Match values that are between 5 and 7 in length
35 # ($m) : Allow ^ on multiline string
36
37 # ----- Back References -----
38 # A back reference allows you to reuse the expression
39 # that proceeds it
40
41 # Grab a double word
42 randStr = "The cat cat fell out the window"
43
44 # Match a word boundary, 1 or more characters followed
45 # by a space if it is then followed by the same
46 # match that is surrounded by the parentheses
47 regex = re.compile(r"(\b\w+)\s+\1")
48
49 matches = re.findall(regex, randStr)
50
51 print("Matches :", len(matches))
52
53 for i in matches:
54     print(i)
55
56 # ----- Back Reference Substitutions -----
57
58 # Replace the bold tags in the link with no tags
59 randStr = "<a href='#'><b>The Link</b></a>"
60
61 # Regex matches bold tags and grabs the text between
62 # them to be used by the back reference
63 regex = re.compile(r"<b>(.*?)</b>")
```

```
64
65 # Replace the tags with just the text between them
66 randStr = re.sub(regex, r"\1", randStr)
67
68 print(randStr)
69
70 # ----- Another Back Reference Substitution -----
71
72 # Receive this string
73 randStr = "412-555-1212"
74
75 # Match the phone number using multiple subexpressions
76 regex = re.compile(r"([\d]{3})-([\d]{3}-[\d]{4})")
77
78 # Output (412)555-1212
79 randStr = re.sub(regex, r"(\1)\2", randStr)
80
81 print(randStr)
82
83 # ----- PROBLEM -----
84 # Receive a string like this
85
86 randStr = "https://www.youtube.com http://www.google.com"
87
88 # Grab the URL and then provide the following output
89 # using a back reference substitution
90 # <a href='https://www.youtube.com'>www.youtube.com</a>
91 # <a href='https://www.google.com'>www.google.com</a>
92
93 regex = re.compile(r"(https?:\/\/([\w.]+))")
94
95 randStr = re.sub(regex, r"<a href='\1'>\2</a>\n", randStr)
96
97 print(randStr)
98
99 # ----- Look Ahead -----
100 # A look ahead defines a pattern to match but not return
101 # You define the expression to look for but not return
102 # like this (?=expression)
103
104 randStr = "One two three four"
105
106 # Grab all letters and numbers of 1 or more separated
107 # by a word boundary but don't include it
108 regex = re.compile(r"\w+(?=\b)")
109
110 matches = re.findall(regex, randStr)
111
112 for i in matches:
113     print(i)
114
115 # ----- Look Behind -----
116 # The look behind looks for what is before the text
117 # to return, but doesn't return it
118 # It is defined like (?<=expression)
119
120 randStr = "1. Bread 2. Apples 3. Lettuce"
121
122 # Find the number, period and space, but only return
123 # the 1 or more letters or numbers that follow
124 regex = re.compile(r"(?<=\d.\s)\w+")
125
126 matches = re.findall(regex, randStr)
127
128 for i in matches:
129     print(i)
130
131 # ----- Look Ahead & Behind -----
132
```

```
133 randStr = "<h1>I'm Important</h1> <h1>So am I</h1>"
134
135 # Use the look behind, get 1 or more of anything,
136 # and use the look ahead
137 regex = re.compile(r"(?<=<h1>).+?(?=</h1>)")
138
139 matches = re.findall(regex, randStr)
140
141 for i in matches:
142     print(i)
143
144 import re
145
146 # ----- Negative Look Ahead & Behind -----
147 # These are used to look for text that doesn't match
148 # the pattern
149
150 # (?!expression) : Negative Look Ahead
151 # (?<!expression) : Negative Look Behind
152
153 randStr = "8 Apples $3, 1 Bread $1, 1 Cereal $4"
154
155 # Grab the total number of grocery items by ignoring the $
156 regex = re.compile(r"(?<!\$)\d+")
157
158 matches = re.findall(regex, randStr)
159
160 print(len(matches))
161
162 # Convert from a string list to an int list
163 matches = [int(i) for i in matches]
164
165 from functools import reduce
166
167 # Sum the items in the list with reduce
168 print("Total Items {}".format(reduce((lambda x, y: x + y), matches)))
```

Learn to Program 18

 Python

```

2
3 # [ ] : Match what is in the brackets
4 # [^ ] : Match anything not in the brackets
5 # ( ) : Return surrounded submatch
6 # . : Match any 1 character or space
7 # + : Match 1 or more of what proceeds
8 # ? : Match 0 or 1
9 # * : Match 0 or More
10 # *? : Lazy match the smallest match
11 # \b : Word boundary
12 # ^ : Beginning of String
13 # $ : End of String
14 # \n : Newline
15 # \d : Any 1 number
16 # \D : Anything but a number
17 # \w : Same as [a-zA-Z0-9_]
18 # \W : Same as [^a-zA-Z0-9_]
19 # \s : Same as [\f\n\r\t\v]
20 # \S : Same as [^\f\n\r\t\v]
21 # {5} : Match 5 of what proceeds the curly brackets
22 # {5,7} : Match values that are between 5 and 7 in length
23 # ($m) : Allow ^ on multiline string
24
25 # Use a back reference to substitute what is between the
26 # bold tags and eliminate the bold tags
27 # re.sub(r"<b>(.*?)</b>", r"\1", randStr)
28
29 # Use a look ahead to find all characters of 1 or more
30 # with a word boundary, but don't return the word
31 # boundary
32 # re.findall(r"\w+(?=\b)", randStr)
33
34 # Use a look behind to find words starting with a number,
35 # period and space, but only return the word that follows
36 # re.findall(r"(?<=\d.\s)\w+", randStr)
37
38 # Use a negative look behind to only return numbers without
39 # a $ in front of them
40 # re.findall(r"(?!\$)\d+", randStr)
41
42 # ----- OR CONDITIONAL -----
43 # You can use | to define the matches you'll except
44
45 randStr = "1. Dog 2. Cat 3. Turtle"
46
47 regex = re.compile(r"\d\.\s(Dog|Cat)")
48
49 matches = re.findall(regex, randStr)
50
51 print(len(matches))
52
53 for i in matches:
54     print(i)
55
56 # ----- PROBLEM -----
57 # Create a regex that will match for 5 digit zip
58 # codes or zip codes with 5 digits a dash and
59 # then 4 digits
60
61 randStr = "12345 12345-1234 1234 12346-333"
62
63 regex = re.compile(r"(\d{5}-\d{4}|\d{5}\s)")
64
65 matches = re.findall(regex, randStr)
66
67 print(len(matches))
68
69 for i in matches:

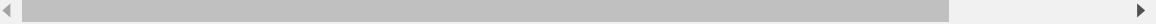
```

```

70     print(i)
71
72     # ----- GROUP -----
73     # We can use group to retrieve parts of regex
74     # matches
75     '''
76     bd = input("Enter your birthday (mm-dd-yyyy) : ")
77
78     bdRegex = re.search(r"(\d{1,2})-(\d{1,2})-(\d{4})", bd)
79
80     print("You were born on", bdRegex.group())
81     print("Birth Month", bdRegex.group(1))
82     print("Birth Day", bdRegex.group(2))
83     print("Birth Year", bdRegex.group(3))
84     '''
85
86     # ----- MATCH OBJECT FUNCTIONS -----
87     # There are functions that provide more information
88     # on your matches
89
90     match = re.search(r"\d{2}", "The chicken weighed 13 lbs")
91
92     # Print the match
93     print("Match :", match.group())
94
95     # Print the start and ending index of the match
96     print("Span :", match.span())
97
98     # Print starting index of the match
99     print("Match :", match.start())
100
101     # Print the ending index of the match
102     print("Match :", match.end())
103
104     # ----- NAMED GROUPS -----
105     # You can also assign names to matches
106
107     randStr = "December 21 1974"
108
109     regex = r"^(?P<month>\w+)\s(?P<day>\d+)\s(?P<year>\d+)"
110
111     matches = re.search(regex, randStr)
112
113     print("Month :", matches.group('month'))
114     print("Day :", matches.group('day'))
115     print("Year :", matches.group('year'))
116
117     # ----- PROBLEM -----
118     # Find all of the following email addresses
119
120     randStr = "d+b@aol.com a_1@yahoo.co.uk A-100@m-b.INTERNATIONAL"
121
122     regex = re.compile(r"[a-zA-Z0-9_+.-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.-.]+")
123
124     matches = re.findall(regex, randStr)
125
126     print(len(matches))
127
128     for i in matches:
129         print(i)
130
131     # ----- PROBLEM -----
132     # Find all of the following phone numbers and then print them
133
134     randStr = "14125551212 4125551212 (412)5551212 412 555 1212 412-555-1212 1-4
135
136     regex = re.compile(r"((1?)(-| ?)(\()?(\\d{3})(\\)|-| |\\)-|\\) )?(\\d{3})(-| )?(\\
137
138     matches = re.findall(regex, randStr)

```

```
139  
140 print(len(matches))  
141  
142 for i in matches:  
143     print(i[0].lstrip())
```



If you value videos like this consider contributing a [\\$1 on Patreon](#).

Code & Transcript

```
Learn to program 19
2  # in Python
3
4  # A database makes it easy for you to organize your
5  # data for storage and fast searching
6
7  # I show how to install SQLite and use it in this
8  # tutorial https://youtu.be/QjICgmk31js
9
10 # You need the SQLite module to use it
11 import sqlite3
12 import sys
13 import csv
14
15 # ----- FUNCTIONS -----
16
17 def printDB():
18     # To retrieve data from a table use SELECT followed
19     # by the items to retrieve and the table to
20     # retrieve from
21
22     try:
23         result = theCursor.execute("SELECT id, FName, LName, Age, Address, Salary, HireDate")
24
25         # You receive a list of lists that hold the result
26         for row in result:
27             print("ID :", row[0])
28             print("FName :", row[1])
29             print("LName :", row[2])
30             print("Age :", row[3])
31             print("Address :", row[4])
32             print("Salary :", row[5])
33             print("HireDate :", row[6])
34
35     except sqlite3.OperationalError:
36         print("The Table Doesn't Exist")
37
38     except:
39         print("Couldn't Retrieve Data From Database")
40
41 # ----- END OF FUNCTIONS -----
42
43 # connect() will open an SQLite database, or if it
44 # doesn't exist it will create it
45 # The file appears in the same directory as this
46 # Python file
47 db_conn = sqlite3.connect('test.db')
48
49 print("Database Created")
50
51 # A cursor is used to traverse the records of a result
52 theCursor = db_conn.cursor()
53
54 # execute() executes a SQL command
55 # We organize our data in tables by defining their
56 # name and the data type for the data
57
58 # We define the table name
59 # A primary key is a unique value that differentiates
60 # each row of data in our table
61 # The primary key will auto increment each time we
62 # add a new Employee
63 # If a piece of data is marked as NOT NULL, that means
```

```
64 # it must have a value to be valid
65
66 # NULL is NULL and stands in for no value
67 # INTEGER is an integer
68 # TEXT is a string of variable length
69 # REAL is a float
70 # BLOB is used to store binary data
71
72 # You can delete a table if it exists like this
73 # db_conn.execute("DROP TABLE IF EXISTS Employees")
74 # db_conn.commit()
75
76 try:
77     db_conn.execute("CREATE TABLE Employees(ID INTEGER PRIMARY KEY AUTOINCRE
78
79     db_conn.commit()
80
81     print("Table Created")
82
83 except sqlite3.OperationalError:
84     print("Table couldn't be Created")
85
86
87 # To insert data into a table we use INSERT INTO
88 # followed by the table name and the item name
89 # and the data to assign to those items
90
91 db_conn.execute("INSERT INTO Employees (FName, LName, Age, Address, Salary,
92                 'VALUES ('Derek', 'Banas', 41, '123 Main St', '500,000', dat
93
94 db_conn.commit()
95
96 print("Employee Entered")
97
98 # Print out all the data in the database
99 printDB()
100
101 # You can update a value in a table by referencing
102 # something unique like the ID or anything else
103 # with the UPDATE command
104
105 try:
106     db_conn.execute("UPDATE Employees SET Address = '121 Main St' WHERE ID=2
107     db_conn.commit()
108
109 except sqlite3.OperationalError:
110     print("Database couldn't be Updated")
111
112 printDB()
113
114 # Delete matching data from the database by
115 # referencing the table name and something unique
116
117 try:
118     db_conn.execute("DELETE FROM Employees WHERE ID=2")
119     db_conn.commit()
120
121 except sqlite3.OperationalError:
122     print("Data couldn't be Deleted")
123
124 printDB()
125
126 # Undo the last commit()
127 db_conn.rollback()
128
129 printDB()
130
131 # You can add a new column to a table with ALTER
132
```

```
133 try:
134     db_conn.execute("ALTER TABLE Employees ADD COLUMN 'Image' BLOB DEFAULT N
135
136     db_conn.commit()
137
138 except sqlite3.OperationalError:
139     print("Table couldn't be Altered")
140
141 # Retrieve table column names
142 theCursor.execute("PRAGMA TABLE_INFO(Employees)")
143
144 # fetchall() returns all remaining rows of a query result
145 # as a list
146 rowNames = [nameTuple[1] for nameTuple in theCursor.fetchall()]
147 print(rowNames)
148
149 # Get the total number of rows
150 theCursor.execute('SELECT COUNT(*) FROM Employees')
151
152 numOfRows = theCursor.fetchall()
153
154 print("Total Rows :", numOfRows[0][0])
155
156 # Get SQLite version
157 theCursor.execute('SELECT SQLITE_VERSION()')
158
159 # fetchone() returns one result
160 print("SQLite Version :", theCursor.fetchone())
161
162 # Use the dictionary cursor to retrieve data in a dictionary
163 with db_conn:
164     db_conn.row_factory = sqlite3.Row
165
166     theCursor = db_conn.cursor()
167
168     theCursor.execute("SELECT * FROM Employees")
169
170     rows = theCursor.fetchall()
171
172     for row in rows:
173         print("{} {}".format(row["FName"], row["LName"]))
174
175 # Write data to File
176
177 with open('dump.sql', 'w') as f:
178
179     # iterdump() returns an iterator to dump the database
180     # in SQL format
181     for line in db_conn.iterdump():
182         f.write('%s\n' % line)
183
184 # Closes the database connection
185 db_conn.close()
```

If you like videos like this consider donating a \$1 on [Patreon](#).

Code & Transcript

```
Learn to Program 20 Python
2  # Interfaces (GUIs) with Tk
3
4  # Tk is a cross platform GUI toolkit that provides a
5  # ton of Widgets (Buttons, Scrollbars, etc.) that are
6  # used to build GUIs
7
8  # TkInter (tee-kay-inter) is included since Python 3.1 on Macs,
9  # Windows and Linux
10
11 # TkInter is a Python interface for Tk
12 from tkinter import *
13 from tkinter import ttk
14
15 # Test to see if TkInter is working
16 # tkinter._test()
17
18 # ----- HELLO WORLD -----
19
20 # root is the main window that surrounds your interface
21 # This creates a Tk object
22 root = Tk()
23
24 # Give your app a title
25 root.title("First GUI")
26
27 # Put a button in the window
28 # Components like button are called Widgets
29 ttk.Button(root, text="Hello TkInter").grid()
30
31 # This keeps the root window visible and your program
32 # running
33 root.mainloop()
34
35
36 # ----- MULTIPLE COMPONENTS -----
37 # Some of the different Widgets : Button, Label,
38 # Canvas, Menu, Text, Scale, OptionMenu, Frame,
39 # CheckButton, LabelFrame, MenuButton, PanedWindow,
40 # Entry, ListBox, Message, RadioButton, ScrollBar,
41 # Bitmap, SpinBox, Image
42
43 root = Tk()
44
45 # Frame widgets surround other widgets
46 frame = Frame(root)
47
48 # We'll use a TkInter variable for our label text
49 # so we can change it with set
50 labelText = StringVar()
51
52 # Create a label and button object
53 # You can set attributes on creation or by calling
54 # methods
55
56 label = Label(frame, textvariable=labelText)
57 button = Button(frame, text="Click Me")
58
59 # Change the label text
60 labelText.set("I am a label")
61
62 # Pack positions the widgets in the window
63 # It is a simple geometry manager
```

```
64 label.pack()
65 button.pack()
66 frame.pack()
67
68 root.mainloop()
69
70
71
72 # ----- PACK GEOMETRY MANAGER -----
73 # Pack positions widgets by allowing them to define
74 # their position (Top, Right, Bottom, Left) and
75 # their fill direction (X, Y, BOTH, NONE) inside
76 # of a box
77
78 root = Tk()
79
80 frame = Frame(root)
81
82 # Define where the widgets should be placed and
83 # how they should be stretched to fill the space
84 Label(frame, text="A Bunch of Buttons").pack()
85 Button(frame, text="B1").pack(side=LEFT, fill=Y)
86 Button(frame, text="B2").pack(side=TOP, fill=X)
87 Button(frame, text="B3").pack(side=RIGHT, fill=X)
88 Button(frame, text="B4").pack(side=LEFT, fill=X)
89
90 frame.pack()
91
92 root.mainloop()
93
94
95 # ----- GRID GEOMETRY MANAGER -----
96 # The Grid manager is the most useful using a series
97 # of rows and columns for laying out widgets
98
99 # Each cell can only hold 1 widget, but a widget
100 # can cover multiple cells.
101
102 root = Tk()
103
104 # rows start at 0, 1, ...
105 # columns start at 0, 1, ...
106 # sticky defines how the widget expands (N, NE, E, SE,
107 # S, SW, W, NW)
108 # padx and pady provide padding around the widget above
109 # and below it
110 Label(root, text="First Name").grid(row=0, sticky=W, padx=4)
111 Entry(root).grid(row=0, column=1, sticky=E, pady=4)
112
113 Label(root, text="Last Name").grid(row=1, sticky=W, padx=4)
114 Entry(root).grid(row=1, column=1, sticky=E, pady=4)
115
116 Button(root, text="Submit").grid(row=3)
117
118 root.mainloop()
119
120
121 # ----- GRID EXAMPLE 2 -----
122
123 root = Tk()
124
125 Label(root, text="Description").grid(row=0, column=0, sticky=W)
126 Entry(root, width=50).grid(row=0, column=1)
127 Button(root, text="Submit").grid(row=0, column=8)
128
129 Label(root, text="Quality").grid(row=1, column=0, sticky=W)
130 Radiobutton(root, text="New", value=1).grid(row=2, column=0, sticky=W)
131 Radiobutton(root, text="Good", value=2).grid(row=3, column=0, sticky=W)
132 Radiobutton(root, text="Poor", value=3).grid(row=4, column=0, sticky=W)
```

```
133 Radiobutton(root, text="Damaged", value=4).grid(row=5, column=0, sticky=W)
134
135 Label(root, text="Benefits").grid(row=1, column=1, sticky=W)
136 Checkbutton(root, text="Free Shipping").grid(row=2, column=1, sticky=W)
137 Checkbutton(root, text="Bonus Gift").grid(row=3, column=1, sticky=W)
138
139 root.mainloop()
140
141 # ----- TKINTER EVENTS -----
142
143 def get_sum(event):
144
145     # Get the value stored in the entries
146     num1 = int(num1Entry.get())
147     num2 = int(num2Entry.get())
148     sum = num1 + num2
149
150     # Delete the value in the entry
151     sumEntry.delete(0, "end")
152
153     # Insert the sum into the entry
154     sumEntry.insert(0, sum)
155
156 root = Tk()
157
158 num1Entry = Entry(root)
159 num1Entry.pack(side=LEFT)
160
161 Label(root, text="+").pack(side=LEFT)
162
163 num2Entry = Entry(root)
164 num2Entry.pack(side=LEFT)
165
166 equalButton = Button(root, text="=")
167
168 # When the left mouse button is clicked call the
169 # function get_sum
170 equalButton.bind("<Button-1>", get_sum)
171
172 equalButton.pack(side=LEFT)
173
174 sumEntry = Entry(root)
175 sumEntry.pack(side=LEFT)
176
177 root.mainloop()
```