1. **Function Basics**  
   Video link [https://vimeo.com/832817302](https://t.dripemail2.com/c/eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkZXRvdXIiLCJpc3MiOiJtb25vbGl0aCIsInN1YiI6ImRldG91cl9saW5rIiwiaWF0IjoxNzU5MjMxNTA4LCJuYmYiOjE3NTkyMzE1MDgsImFjY291bnRfaWQiOiI0MjU0NDk3IiwiZGVsaXZlcnlfaWQiOiJ0OGFkdXVsdG8wZjIzcGl5ZThpdCIsInRva2VuIjoidDhhZHV1bHRvMGYyM3BpeWU4aXQiLCJzZW5kX2F0IjoxNzU5MjQ0NDAwLCJlbWFpbF9pZCI6NzUwNDk2NiwiZW1haWxhYmxlX3R5cGUiOiJDYW1wYWlnbiIsImVtYWlsYWJsZV9pZCI6NDc4NjE0LCJ1cmwiOiJodHRwczovL3ZpbWVvLmNvbS84MzI4MTczMDI_X19zPW16cG02bnEzY210engwM3RrNWRnIn0.bLfr0hXOPHOC0g8sj8hgEls8jzOPmsfhW6SWjaITqlg)  
   Length is 4 minutes
2. **Positional and Named Arguments**  
   Video link [https://vimeo.com/839580261](https://t.dripemail2.com/c/eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkZXRvdXIiLCJpc3MiOiJtb25vbGl0aCIsInN1YiI6ImRldG91cl9saW5rIiwiaWF0IjoxNzU5MjMxNTA4LCJuYmYiOjE3NTkyMzE1MDgsImFjY291bnRfaWQiOiI0MjU0NDk3IiwiZGVsaXZlcnlfaWQiOiJ0OGFkdXVsdG8wZjIzcGl5ZThpdCIsInRva2VuIjoidDhhZHV1bHRvMGYyM3BpeWU4aXQiLCJzZW5kX2F0IjoxNzU5MjQ0NDAwLCJlbWFpbF9pZCI6NzUwNDk2NiwiZW1haWxhYmxlX3R5cGUiOiJDYW1wYWlnbiIsImVtYWlsYWJsZV9pZCI6NDc4NjE0LCJ1cmwiOiJodHRwczovL3ZpbWVvLmNvbS84Mzk1ODAyNjE_X19zPW16cG02bnEzY210engwM3RrNWRnIn0.T0Djlj-Aa7DDLEHruJXFCa4Vmm-yWGeGBCr2I8SR6GM)  
   Length is 6 minutes
3. **Default Values**  
   Video link [https://vimeo.com/833453159](https://t.dripemail2.com/c/eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkZXRvdXIiLCJpc3MiOiJtb25vbGl0aCIsInN1YiI6ImRldG91cl9saW5rIiwiaWF0IjoxNzU5MjMxNTA4LCJuYmYiOjE3NTkyMzE1MDgsImFjY291bnRfaWQiOiI0MjU0NDk3IiwiZGVsaXZlcnlfaWQiOiJ0OGFkdXVsdG8wZjIzcGl5ZThpdCIsInRva2VuIjoidDhhZHV1bHRvMGYyM3BpeWU4aXQiLCJzZW5kX2F0IjoxNzU5MjQ0NDAwLCJlbWFpbF9pZCI6NzUwNDk2NiwiZW1haWxhYmxlX3R5cGUiOiJDYW1wYWlnbiIsImVtYWlsYWJsZV9pZCI6NDc4NjE0LCJ1cmwiOiJodHRwczovL3ZpbWVvLmNvbS84MzM0NTMxNTk_X19zPW16cG02bnEzY210engwM3RrNWRnIn0.1bN4_9keSWJmNZ7aWqNJWJ04N9sarwvZHDRB-O6eVf8)  
   Length is 1 minute
4. **Return Values**  
   Video link  [https://vimeo.com/833454652](https://t.dripemail2.com/c/eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkZXRvdXIiLCJpc3MiOiJtb25vbGl0aCIsInN1YiI6ImRldG91cl9saW5rIiwiaWF0IjoxNzU5MjMxNTA4LCJuYmYiOjE3NTkyMzE1MDgsImFjY291bnRfaWQiOiI0MjU0NDk3IiwiZGVsaXZlcnlfaWQiOiJ0OGFkdXVsdG8wZjIzcGl5ZThpdCIsInRva2VuIjoidDhhZHV1bHRvMGYyM3BpeWU4aXQiLCJzZW5kX2F0IjoxNzU5MjQ0NDAwLCJlbWFpbF9pZCI6NzUwNDk2NiwiZW1haWxhYmxlX3R5cGUiOiJDYW1wYWlnbiIsImVtYWlsYWJsZV9pZCI6NDc4NjE0LCJ1cmwiOiJodHRwczovL3ZpbWVvLmNvbS84MzM0NTQ2NTI_X19zPW16cG02bnEzY210engwM3RrNWRnIn0.gfTxKwQ-hedi7ZwuMLhWFN3M1meMiuoZmkAdJapmv2U)  
   Length is 1 minute
5. **LEGB Rules - Resolving Names**  
   Video link [https://vimeo.com/833466375](https://t.dripemail2.com/c/eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkZXRvdXIiLCJpc3MiOiJtb25vbGl0aCIsInN1YiI6ImRldG91cl9saW5rIiwiaWF0IjoxNzU5MjMxNTA4LCJuYmYiOjE3NTkyMzE1MDgsImFjY291bnRfaWQiOiI0MjU0NDk3IiwiZGVsaXZlcnlfaWQiOiJ0OGFkdXVsdG8wZjIzcGl5ZThpdCIsInRva2VuIjoidDhhZHV1bHRvMGYyM3BpeWU4aXQiLCJzZW5kX2F0IjoxNzU5MjQ0NDAwLCJlbWFpbF9pZCI6NzUwNDk2NiwiZW1haWxhYmxlX3R5cGUiOiJDYW1wYWlnbiIsImVtYWlsYWJsZV9pZCI6NDc4NjE0LCJ1cmwiOiJodHRwczovL3ZpbWVvLmNvbS84MzM0NjYzNzU_X19zPW16cG02bnEzY210engwM3RrNWRnIn0.PCvDajPIlL0uhyJoUUrSYR4xTIJ8DiohwE0OCDxztrE)  
   Length is 5 minutes
6. **Additional Aspects of Functions**  
   Video link [https://vimeo.com/833468631](https://t.dripemail2.com/c/eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkZXRvdXIiLCJpc3MiOiJtb25vbGl0aCIsInN1YiI6ImRldG91cl9saW5rIiwiaWF0IjoxNzU5MjMxNTA4LCJuYmYiOjE3NTkyMzE1MDgsImFjY291bnRfaWQiOiI0MjU0NDk3IiwiZGVsaXZlcnlfaWQiOiJ0OGFkdXVsdG8wZjIzcGl5ZThpdCIsInRva2VuIjoidDhhZHV1bHRvMGYyM3BpeWU4aXQiLCJzZW5kX2F0IjoxNzU5MjQ0NDAwLCJlbWFpbF9pZCI6NzUwNDk2NiwiZW1haWxhYmxlX3R5cGUiOiJDYW1wYWlnbiIsImVtYWlsYWJsZV9pZCI6NDc4NjE0LCJ1cmwiOiJodHRwczovL3ZpbWVvLmNvbS84MzM0Njg2MzE_X19zPW16cG02bnEzY210engwM3RrNWRnIn0.H-_U481P4rpOKsC0SfqS_oZ2Se5PH0-hmLovoA88Urs)  
   Length is 3 minutes
7. **Using \*args to Pass in Arguments** (\*optional/intermediate content)  
   Video link [https://vimeo.com/833474434](https://t.dripemail2.com/c/eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkZXRvdXIiLCJpc3MiOiJtb25vbGl0aCIsInN1YiI6ImRldG91cl9saW5rIiwiaWF0IjoxNzU5MjMxNTA4LCJuYmYiOjE3NTkyMzE1MDgsImFjY291bnRfaWQiOiI0MjU0NDk3IiwiZGVsaXZlcnlfaWQiOiJ0OGFkdXVsdG8wZjIzcGl5ZThpdCIsInRva2VuIjoidDhhZHV1bHRvMGYyM3BpeWU4aXQiLCJzZW5kX2F0IjoxNzU5MjQ0NDAwLCJlbWFpbF9pZCI6NzUwNDk2NiwiZW1haWxhYmxlX3R5cGUiOiJDYW1wYWlnbiIsImVtYWlsYWJsZV9pZCI6NDc4NjE0LCJ1cmwiOiJodHRwczovL3ZpbWVvLmNvbS84MzM0NzQ0MzQ_X19zPW16cG02bnEzY210engwM3RrNWRnIn0.lwJarVw_ePn_fkmRg5qqxSuMXfZ_8dbfFcur0mdf4q8)  
   Length is 4 minutes
8. **Using \*\*kwargs to Pass in Keyword Arguments** (\*optional/intermediate content)  
   Video link [https://vimeo.com/833477971](https://t.dripemail2.com/c/eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkZXRvdXIiLCJpc3MiOiJtb25vbGl0aCIsInN1YiI6ImRldG91cl9saW5rIiwiaWF0IjoxNzU5MjMxNTA4LCJuYmYiOjE3NTkyMzE1MDgsImFjY291bnRfaWQiOiI0MjU0NDk3IiwiZGVsaXZlcnlfaWQiOiJ0OGFkdXVsdG8wZjIzcGl5ZThpdCIsInRva2VuIjoidDhhZHV1bHRvMGYyM3BpeWU4aXQiLCJzZW5kX2F0IjoxNzU5MjQ0NDAwLCJlbWFpbF9pZCI6NzUwNDk2NiwiZW1haWxhYmxlX3R5cGUiOiJDYW1wYWlnbiIsImVtYWlsYWJsZV9pZCI6NDc4NjE0LCJ1cmwiOiJodHRwczovL3ZpbWVvLmNvbS84MzM0Nzc5NzE_X19zPW16cG02bnEzY210engwM3RrNWRnIn0.yMItnxcrc0tciwoMsgzyvJny1dvAD32qG4C0B_m6sLg)  
   Length is 4 minutes
9. **Using \*args and \*\*kwargs in Function Definitions** (\*optional/intermediate content)  
   Video link  [https://vimeo.com/833481957](https://t.dripemail2.com/c/eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkZXRvdXIiLCJpc3MiOiJtb25vbGl0aCIsInN1YiI6ImRldG91cl9saW5rIiwiaWF0IjoxNzU5MjMxNTA4LCJuYmYiOjE3NTkyMzE1MDgsImFjY291bnRfaWQiOiI0MjU0NDk3IiwiZGVsaXZlcnlfaWQiOiJ0OGFkdXVsdG8wZjIzcGl5ZThpdCIsInRva2VuIjoidDhhZHV1bHRvMGYyM3BpeWU4aXQiLCJzZW5kX2F0IjoxNzU5MjQ0NDAwLCJlbWFpbF9pZCI6NzUwNDk2NiwiZW1haWxhYmxlX3R5cGUiOiJDYW1wYWlnbiIsImVtYWlsYWJsZV9pZCI6NDc4NjE0LCJ1cmwiOiJodHRwczovL3ZpbWVvLmNvbS84MzM0ODE5NTc_X19zPW16cG02bnEzY210engwM3RrNWRnIn0.bGKez6GGBrZ6Ve8SiXgJTK8Nr_IDt4-q1YlGgaTDgRo)  
   Length is 3 minutes
10. **Lambda Expressions** (\*optional/intermediate content)  
    Video link [https://vimeo.com/845996047](https://t.dripemail2.com/c/eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkZXRvdXIiLCJpc3MiOiJtb25vbGl0aCIsInN1YiI6ImRldG91cl9saW5rIiwiaWF0IjoxNzU5MjMxNTA4LCJuYmYiOjE3NTkyMzE1MDgsImFjY291bnRfaWQiOiI0MjU0NDk3IiwiZGVsaXZlcnlfaWQiOiJ0OGFkdXVsdG8wZjIzcGl5ZThpdCIsInRva2VuIjoidDhhZHV1bHRvMGYyM3BpeWU4aXQiLCJzZW5kX2F0IjoxNzU5MjQ0NDAwLCJlbWFpbF9pZCI6NzUwNDk2NiwiZW1haWxhYmxlX3R5cGUiOiJDYW1wYWlnbiIsImVtYWlsYWJsZV9pZCI6NDc4NjE0LCJ1cmwiOiJodHRwczovL3ZpbWVvLmNvbS84NDU5OTYwNDc_X19zPW16cG02bnEzY210engwM3RrNWRnIn0.EtjNS4oZwd2ZO0jEepgYOYC2LKOj3Gc7qjLrz6QoHCc)  
    Length is 8 minutes

**\* Optional/intermediate content - this content is more intermediate and can be safely skipped.**In other words, feel free to skip this content if you are time limited or are struggling with the more fundamental content.  
  
  
  
**Collateral Material (programs used in the videos):**  
  
*In collateral, I try to provide the Python programs shown in the videos. In other words, I try to make it easy for you to reproduce the examples from the videos.*  
  
[Lesson6 Collateral Programs](https://t.dripemail2.com/c/eyJhbGciOiJIUzI1NiJ9..osY0xdqiaTl0IYHEDRNIl7N3D9BjSB3za38uqYziFRg)  
  
  
  
**Additional Content:**  
  
[How to Define Functions in Python 3](https://t.dripemail2.com/c/eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkZXRvdXIiLCJpc3MiOiJtb25vbGl0aCIsInN1YiI6ImRldG91cl9saW5rIiwiaWF0IjoxNzU5MjMxNTA4LCJuYmYiOjE3NTkyMzE1MDgsImFjY291bnRfaWQiOiI0MjU0NDk3IiwiZGVsaXZlcnlfaWQiOiJ0OGFkdXVsdG8wZjIzcGl5ZThpdCIsInRva2VuIjoidDhhZHV1bHRvMGYyM3BpeWU4aXQiLCJzZW5kX2F0IjoxNzU5MjQ0NDAwLCJlbWFpbF9pZCI6NzUwNDk2NiwiZW1haWxhYmxlX3R5cGUiOiJDYW1wYWlnbiIsImVtYWlsYWJsZV9pZCI6NDc4NjE0LCJ1cmwiOiJodHRwczovL3d3dy5kaWdpdGFsb2NlYW4uY29tL2NvbW11bml0eS90dXRvcmlhbHMvaG93LXRvLWRlZmluZS1mdW5jdGlvbnMtaW4tcHl0aG9uLTM_X19zPW16cG02bnEzY210engwM3RrNWRnIn0.WS2hoG1wwT3TyQn_m24UjkqX7Xs2dKt4bw05Iyu-Pqw)  
Digital Ocean article on functions. This article covers function syntax and creation, parameters, arguments, using positional arguments, using keyword arguments, default values, and returning values from a function. The article has a section on using a 'main()' function and on the \_\_name\_\_ == "\_\_main\_\_" technique which was fairly confusing (so I would just skip that section and learn that content elsewhere).  
  
  
[Defining Your Own Python Function](https://t.dripemail2.com/c/eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkZXRvdXIiLCJpc3MiOiJtb25vbGl0aCIsInN1YiI6ImRldG91cl9saW5rIiwiaWF0IjoxNzU5MjMxNTA4LCJuYmYiOjE3NTkyMzE1MDgsImFjY291bnRfaWQiOiI0MjU0NDk3IiwiZGVsaXZlcnlfaWQiOiJ0OGFkdXVsdG8wZjIzcGl5ZThpdCIsInRva2VuIjoidDhhZHV1bHRvMGYyM3BpeWU4aXQiLCJzZW5kX2F0IjoxNzU5MjQ0NDAwLCJlbWFpbF9pZCI6NzUwNDk2NiwiZW1haWxhYmxlX3R5cGUiOiJDYW1wYWlnbiIsImVtYWlsYWJsZV9pZCI6NDc4NjE0LCJ1cmwiOiJodHRwczovL3JlYWxweXRob24uY29tL2RlZmluaW5nLXlvdXItb3duLXB5dGhvbi1mdW5jdGlvbi8_X19zPW16cG02bnEzY210engwM3RrNWRnIn0.6HHNhFH7eJnXAgqng_-M0ra7aE9vZPMMQ2eeiXE6tPQ)  
Exceptionally good article from the people at Real Python that covers various aspects of Python functions. It talks about the reasons and advantages of using functions, function syntax, positional arguments, named arguments, mixing positional and named arguments, default parameters, the strange behavior of mutable default parameters. The article also discusses pass-by-value and pass-by-reference versus what Python does and how that affects function behavior. The article talks about function side-effects for mutable arguments. Finally, it covers various uses of \*args, and \*\*kwargs. There is a section on 'Python Function Annotations' that you can safely skip (this pertains to Python type hints which we have not covered in this course).  
  
  
  
  
**Exercises**  
  
Reference code for these exercises is posted on GitHub at:  
    [https://github.com/twin-bridges/learning\_python/tree/main/lesson6/exercises](https://t.dripemail2.com/c/eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkZXRvdXIiLCJpc3MiOiJtb25vbGl0aCIsInN1YiI6ImRldG91cl9saW5rIiwiaWF0IjoxNzU5MjMxNTA4LCJuYmYiOjE3NTkyMzE1MDgsImFjY291bnRfaWQiOiI0MjU0NDk3IiwiZGVsaXZlcnlfaWQiOiJ0OGFkdXVsdG8wZjIzcGl5ZThpdCIsInRva2VuIjoidDhhZHV1bHRvMGYyM3BpeWU4aXQiLCJzZW5kX2F0IjoxNzU5MjQ0NDAwLCJlbWFpbF9pZCI6NzUwNDk2NiwiZW1haWxhYmxlX3R5cGUiOiJDYW1wYWlnbiIsImVtYWlsYWJsZV9pZCI6NDc4NjE0LCJ1cmwiOiJodHRwczovL2dpdGh1Yi5jb20vdHdpbi1icmlkZ2VzL2xlYXJuaW5nX3B5dGhvbi90cmVlL21haW4vbGVzc29uNi9leGVyY2lzZXM_X19zPW16cG02bnEzY210engwM3RrNWRnIn0.Pd1onRqHELIa8zeXYsdNRTN2BmRKm4DvUkEeu7a3GE4)  
  
***Note, you will need some sort of a network device to work on these exercises. This can be a virtual or physical device. Make sure you are only working on test or lab devices.***  
  
***If you do not want to use telnet and want to use SSH instead, you can do exercise1, exercise2, and exercise5 using an SSH solution.****The authentication process for SSH is generally done at a lower level (so you don't have to handle the username and password prompting). Because of this, exercise1 becomes create a read() function that reads the initial device prompt, exercise2 becomes create a write() function and test by writing out a newline, and exercise3/exercise4 become not applicable. Exercise5 should be fine as originally stated.*  
  
*For reference on the SSH glue code to use see:*  
  
[https://github.com/twin-bridges/learning\_python/tree/main/lesson6/collateral/SSH](https://t.dripemail2.com/c/eyJhbGciOiJIUzI1NiJ9..2fFcDvvcSg6Dlf1UD6UsHwOdLL2Ae_-enxKF4FITDmA)  
  
[https://github.com/twin-bridges/learning\_python/tree/main/lesson6/exercises/SSH](https://t.dripemail2.com/c/eyJhbGciOiJIUzI1NiJ9..7h_41oki9DWsyD2qhzfD37oxeZfGOdVAy18H7T8ItVw)  
  
  
  
1. You can start a simple telnet connection using the following code:

from telnetlib import Telnet

import time

host = "[device.domain.com](http://device.domain.com/)"

username = "admin"

# Create telnet connection

tn = Telnet(host)

# Wait for the device to respond

time.sleep(1.5)

data = tn.read\_very\_eager()

print(data)​

If connecting to a Cisco router your output will likely be similar to the following:

b'\r\n\r\nUser Access Verification\r\n\r\nUsername: '

Note, this is a byte-string and we can convert it to a unicode string by calling the .decode() method on the byte-string.  
  
Using the above, create a ***read()****f*unction that takes two arguments: telnet\_conn and sleep. Default the sleep argument to 1.5 seconds. In this function perform the sleep, telnet channel read, byte-string decoding, and return the read data.  
  
Your new telnet code should now look as follows (intentionally not showing the read() function that you are supposed to create).

host = "[device.domain.com](http://device.domain.com/)"

username = "admin"

# Create telnet connection

tn = Telnet(host)

data = read(tn)

print(data)

2. At this point, we are telnetting but have not fully logged in. In order to fully login, we need to be able to handle the username/password prompting. For my Cisco IOS device this looks like the following:

$ telnet [cisco1.lasthop.io](http://cisco1.lasthop.io/)

Trying 184.105.247.70...

Connected to [cisco1.lasthop.io](http://cisco1.lasthop.io/).

Escape character is '^]'.

User Access Verification

Username: pyclass

Password:

cisco1#

In order to handle the Username prompt we need to write data to the channel. This can be accomplished by using the telnetlib .write() method. The .write() method takes a byte-string.  
  
So we need to search for "Username" in the data that we read and then send "admin\n" (don't forget the <enter> at the end). We need to send this username as a byte-string.  
  
In order to send it as a byte-string, we simply call the .encode() method on our unicode string. This will convert our internal string to a UTF-8 encoded byte-string (think of UTF-8 encoding as representing our internal string in a certain standardized way to the remote device).  
  
Create a new 'write' function that takes in both the telnet connection and also some data (i.e. the data that you will send to the remote device). The function should convert this data to UTF-8 (i.e. a UTF-8 encoded byte string) and send it down the telnet channel.  
  
Now that you have both a read() function and a write() function. Use these two functions to read in the initial data, perform a regular expression search on that data looking for the Username (or whatever your test device presents here) and then writes out the username ("admin\n" in my example).  
  
Perform one final read() and verify that you receive the "Password" prompt back (or whatever the equivalent is on your test device).  
  
  
3. Expand on exercise2 except use your read() and write() functions to properly handle both the username prompting and the password prompting. Use Python's getpass() to have the script prompt for the password.  
  
Perform one final read() and verify that you are fully logged in to the device.  
  
  
4. Modify the code that you created in exercise2 and exercise3 and create a login() function.  
  
This login() function should handle both sending the username and sending the password. Your code should use regular expressions to detect both the username and the password prompt.  
  
After sending the password, your code should verify that you have fully logged in successfully. In other words, you should have a regular expression search that matches the network device's prompt. Your login() function should return a boolean indicating that it detected the network device's prompt (True) or that it failed to do so (False).  
  
Your login() function should accept three arguments: the telnet\_connection, the username, and the password. In your test code, pass these three arguments in as named arguments.  
  
  
5. Create a show\_cmd() function that sends a command down the telnet channel and reads back the command output.  
  
The function should take two arguments: telnet\_conn and cmd. The cmd parameter should have a default value of "show ip interface brief" (or some similar command that works on your test device).  
  
Have your function return the output from this command.  
  
Back in your main program, call your show\_cmd() function using both the default 'cmd' and then again with a separate 'cmd' of your choice. Pass the telnet\_connection in as a positional argument and the cmd in as a named argument (this assumes that telnet\_conn is the first parameter in the function definition).

**CLASS OUTLINE**

1. **Function Basics**
   1. Functions let you have reusable building blocks [0:24]
   2. Code Reusability - code can solve a problem and create building blocks to solve numerous problems [0:58]
   3. Functions - Why? Write something once and use multiple times [1:39]
   4. Functions Syntax [1:41]
      1. def - function definition [1:44]
      2. Parameters - variable(s) local to the function [2:01]
      3. Indented code block [2:44]
      4. Return statement (implicit return none) [2:56]
   5. Call function and specify arguments to pass in [3:23]
2. **Positional and Named Arguments**
   1. Parameter - variable inside the parentheses in a function definition (only exists in the function)  [0:10]
   2. Arguments - things that are passed in to the function call [0:17]
   3. Positional Arguments - bound to the position order in the function, first-to-first; second-to-second [0:42]
   4. Calling Functions: Using Named Arguments - explicitly telling Python via naming the arguments, position no longer matters [2:08]
   5. Mix and match positional and named arguments - Python says this is fine, as long as you pass all of your positional arguments first [3:20]
   6. Example. If there are no parameters available, there will be an error message [4:58]
3. **Default Values**
   1. Can give parameters a default value and it no longer needs to be included in the function call [0:19]
      1. A value specified in your function call would override the default value [0:48]
   2. Note: Default values is very useful for expanding function across time [0:57]
4. **Return Values**
   1. Functions can return results which can then be used outside of the function [0:14]
5. **LEGB Rules - Resolving Names**
   1. Where do functions look for variables (LEGB Rule) [0:10]
      1. L = Local - looks locally for defined variable [1:01]
      2. E = Enclosed (nested inside other functions) [1:42]
      3. G = Global namespace, in the main program [2:09]
      4. B = Builtins (and keywords) - things that are part of the Python language itself [2:41]
   2. You can use linters to make sure you’re not overriding any builtins [3:45]
   3. Note, if you have a variable(s) in the global namespace, that would be distinct from the local variable even if they have the same name  [3:58]
6. **Additional Aspects of Functions**
   1. Functions: Things that might not be obvious [0:07]
      1. Function parameters/arguments don’t have to be strings, can be numbers, lists, and dictionaries [0:11]
      2. Functions always return something [0:50]
      3. Do not directly use lists or dictionaries with default values. [1:01]
         1. Lists and dictionaries are mutable  [1:29]
         2. Why? Python sets this variable when it parses it and doesn’t always return the value to be none  [2:06]
7. **Using Star-args to Pass in Arguments (\*optional/intermediate content)**
   1. Is there a way we can pass a list of arguments? [0:50]
   2. What if we directly try to pass a list (containing two elements)? We get an error because the entire list gets passed as the first element and the second element is not [1:28]
   3. Passing argument using \*args [2:15]
      1. By putting a \* in front of a list or tuple we tell the function to pass in the elements of the list/tuple (in order) [2:30]
         1. Correspondence between the number of elements in the list and number of parameters in our function definition [3:10]
8. **Using Star-star-kwargs to Pass in Keyword Arguments (\*optional/intermediate content)**
   1. What if we want to pass keyword-arguments? [0:09]
      1. Example: straight pass the dictionary to the function positionally (gets and error, binds to first parameter [0:45]
      2. Example: \*arg (single \*) pass in just the dictionary keys positionally [1:15]
      3. Example: \*\*kwargs (double \*\*) pass in as key-value pairs [2:22]
      4. \*\*kwargs - pass the dictionary in as key value pairs (keys must exactly match the parameter names.) [2:57]
9. **Using Star-args and Star-star-kwargs in Function Definitions (\*optional/intermediate content)**
   1. What if we have we have \*args in the function definition? [0:15]
   2. Python has a way to prefix the parameters in the function definition with star [0:36]
      1. This allows us to take in any number of positional arguments [0:49]
      2. Example:  Call function that is prefixed with a star. Python will create a tuple in the order the elements were passed in [1:18]
   3. And what about accepting any phone of keyword arguments (in the function definition)? [2:00]
      1. This allows us to take in any number of keyword arguments [2:07]
      2. Example: Call function that is prefixed with star star. Python will create a dictionary with all the keyword arguments  [2:40]
10. **Lambda Expressions**
    1. What are they? Single line functions with constraints [0:19]
    2. Syntax [0:34]
       1. Argument(s)/ variable that are passed in [0:40]
       2. What gets returned from the lambda is after the colon and always has to be an expression [0:50]
       3. Lambda is an anonymous function, there is not inherent name [1:42]
       4. Variables are wrapped in parentheses and there is an implicit return statement [2:12]
       5. Multiple parameters can be passed in [2:25]
    3. How can we use Lambda Expressions? [2:42]
       1. Binding to a variable and calling the “function” [2:50]
       2. You could also call this “function” using named arguments [3:18]
    4. Common uses
       1. Sorting [3:37]
       2. Overriding the default sorting method using the “key” argument [4:12]
       3. Sorting lists of tuples [6:02]  
          ​