Python Course notes

Wednesday, June 12, 2024 10:00 AM

06/13/2024 - Module 1

https://docs.python.org/3/library/functions.html

https://www.w3schools.com/python/python_ref_keywords.asp

https://flexiple.com/python/arithmetic-operators-in-python/

2**3 - power of example

Values: True, False, None
Conditions: if, elif, else
Logical operators: and, or, not
Loops: for, in, while, break, continue
Functions: def, return

Arithmetic operators
Python can calculate numbers using common mathematical operators, along with some special operators, too:

x + y	Addition + operator returns the sum of x plus y
x - y	Subtraction - operator returns the difference of x minus y
x * y	Multiplication * operator returns the product of x times y
x / y	Division / operator returns the quotient of x divided by y
x**y	Exponent ** operator returns the result of raising x to the power of y
x**2	Square expression returns x squared
x**3	Cube expression returns x cubed
x**(1/2)	Square root (½) or (0.5) fractional exponent operator returns the square root of x
x // y	Floor division operator returns the integer part of the integer division of x by y
x % y	Modulo operator returns the remainder part of the integer division of x by y

Order of operations
The order of operations are to be calculated from left to right in the following order:

Parentheses ( ), { }, [ ]

Exponents xy (x**y)

Multiplication * and Division /

Addition + and Subtraction -

You might find the PEMDAS mnemonic device to be helpful in remembering the order.

https://docs.python.org/3/library/functions.html - Lists and summarizes Python's built-in functions.

https://docs.python.org/3/library/functions.html - Lists Python's reserved keywords and a brief description of what each keyword does.

Terms and definitions from Course 1, Module 1
Automation: The process of replacing a manual step with one that happens automatically

Client-side scripting language: Primarily for web programming; the scripts are transferred from a web server to the end-user's internet browser, then executed in the browser

Code editors: Tools to provide features, including syntax highlighting, automatic indentation, error checking, and autocompletion

Computer program: A step-by-step list of instructions that a computer follows to reach an intended goal

Functions: A reusable block of code that performs a specific task

IDE: A software application that provides comprehensive facilities for software development

Interpreter: The program that reads and executes code

Input: Information that is provided to a program by the end user

Logic errors: Errors in code that prevent it from running correctly

Machine language: Lowest-level computer language. It communicates directly with computing machines in binary code (ones and zeros)

Object-oriented programming language: Most coding elements are considered to be objects with configurable properties

Output: the end result of a task performed by a function or computer program

Platform-specific scripting language: Language used by system administrators on those specific platforms

Programming: The process of writing a program to behave in different ways

Programming code: A set of written computer instructions, guided by rules, using a computer programming language

Programming languages: Language with syntax and semantics to write computer programs

Python: A general purpose programming language

Python interpreter: Program that reads and executes Python code by translating Python code into computer instructions

Script: Often used to automate specific tasks

Semantics: The intended meaning or effect of statements, or collections of words, in both human and computer languages

Syntax: The rules for how each statements are constructed in both human and computer languages

Variables: These are used to temporarily store changeable values in programming code

06/14/2024 - Module 2

Naming rules and conventions
When assigning names to objects, programmers adhere to a set of rules and conventions which help to standardize code and make it more accessible to everyone. Here are some naming rules and conventions that you should know:

Names cannot contain spaces.

Names may be a mixture of upper and lower case characters.

Names can’t start with a number but may contain numbers after the first character.

Variable names and function names should be written in snake_case, which means that all letters are lowercase and words are separated using an underscore.

Descriptive names are better than cryptic abbreviations because they help other programmers (and you) read and interpret your code. For example, student_name is better than sn. It may feel excessive when you write it, but when you return to your code you’ll find it much easier to understand.

Tim Peters, a Python programmer, wrote this now-famous “poem” of guiding principles for coding in Python:

The Zen of Python

Beautiful is better than ugly.
Explicit is better than implicit.

Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one—and preferably only one—obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

Implicit vs explicit conversion
As we saw earlier in the video, some data types can be mixed and matched due to implicit conversion. Implicit conversion is where the interpreter helps us out and automatically converts one data type into another, without having to explicitly tell it to do so.

By contrast, explicit conversion is where we manually convert from one data type to another by calling the relevant function for the data type we want to convert to. We used this in our video example when we wanted to print a number alongside some text. Before we could do that, we needed to call the str() function to convert the number into a string. Once the number was explicitly converted to a string, we could join it with the rest of our textual string and print the result.

Terms
expression - a combination of numbers, symbols, or other values that produce a result when evaluated

data types - classes of data (e.g., string, int, float, Boolean, etc.), which include the properties and behaviors of instances of the data type (variables)

variable - an instance of a data type class, represented by a unique name within the code, that stores changeable values of the specific data type

implicit conversion - when the Python interpreter automatically converts one data type to another

explicit conversion - when code is written to manually convert one data type to another using a data type conversion function:

str() - converts a value (often numeric) to a string data type

int() - converts a value (usually a float) to an integer data type

float() - converts a value (usually an integer) to a float data type

print()
The print() function outputs a specified object to the screen. The print() function is one of the most commonly used functions in Python because it allows you to output any detail from your code.

To use the print() function, you pass the object you want to print as an argument to the function. The print() function takes in any number of arguments, separated by a comma, and prints all of them. For example, you can run the following code that prints a string, a variable, another string, and an integer together:

12
```
month = "September"
print("Investigate failed login attempts during", month, "if more than", 100)
```
Reset
```
Investigate failed login attempts during September if more than 100
```

type()

The type() function returns the data type of its argument. The type() function helps you keep track of the data types of variables to avoid errors throughout your code.

To use it, you pass the object as an argument, and it returns its data type. It only accepts one argument. For example, you could specify type("security") or type(7).

Passing one function into another
When working with functions, you often need to pass them through print() if you want to output the data type to the screen. This is the case when using a function like type(). Consider the following code:

```
print(type("This is a string"))
```
Reset
It displays str(), which means that the argument passed to the type() function is a string. This happens because the type() function is processed first and its output is passed as an argument to the print() function.

str()
The str() function can be used to convert any data type to a string. The str()function takes a single argument, which is the value that you want to convert to a string. The str() function will then return a string representation of the value.

In this example, the str() function will convert the number 12 to a string. This will run the code and print the string representation of the number.

```
number = 12
string_representation = str(number)
print(string_representation)
```
Reset

sorted()
The sorted() function sorts the components of a list. The sorted() function also works on any iterable, like a string, and returns the sorted elements in a list. By default, it sorts them in ascending order. When given an iterable that contains numbers, it sorts them from smallest to largest; this includes iterables that contain numeric data as well as iterables that contain string data beginning with numbers. An iterable that contains strings that begin with alphabetic characters will be sorted alphabetically.

The sorted() function takes an iterable, like a list or a string, as an input. So, for example, you can use the following code to sort the list of login sessions from shortest to longest:

```
time_list = [12, 2, 32, 19, 57, 22, 14]
print(sorted(time_list))
```
Reset
This displays the sorted list.

The sorted() function does not change the iterable that it sorts. The following code illustrates this:

```
time_list = [12, 2, 32, 19, 57, 22, 14]
print(sorted(time_list))
print(time_list)
```
Reset
The first print() function displays the sorted list. However, the second print() function, which does not include the sorted() function, displays the list as assigned to time_list in the first line of code.

One more important detail about the sorted() function is that it cannot take lists or strings that have elements of more than one data type. For example, you can’t use the list [1, 2, "hello"].

max() and min()
The max() function returns the largest numeric input passed into it. The min() function returns the smallest numeric input passed into it.

The max() and min() functions accept arguments of either multiple numeric values or of an iterable like a list, and they return the largest or smallest value respectively.

For example, you could use these functions to identify the longest or shortest session that a user logged in for. If a specific user logged in seven times during a week, and you stored their access times in minutes in a list, you can use the max() and min() functions to find and print their longest and shortest sessions:

123
```
time_list = [12, 2, 32, 19, 57, 22, 14]
print(min(time_list))
print(max(time_list))
```
Reset

Key takeaways
Built-in functions are powerful tools in Python that allow you to perform tasks with one simple command. The print() function prints its arguments to the screen, the type() function returns the data type of its argument, the sorted() organizes its argument, and the min() and max() functions return the smallest and largest values of an iterable respectively.

Len() - length of a string

Study guide: Functions
This study guide provides a quick-reference summary of what you learned in this lesson and serves as a guide for the upcoming practice quiz.

In the Functions segment, you learned how to define and call functions, utilize a function's parameters, and return data from a function. You also learned how to differentiate and convert between different data types utilizing variables. Plus, you learned a few best practices for writing reusable and readable code.

Terms
return value - the value or variable returned as the end result of a function

parameter (argument) - a value passed into a function for use within the function

refactoring code - a process to restructure code without changing functionality

Knowledge
The purpose of the def() keyword is to define a new function.

Best practices for writing code that is readable and reusable:

Create a reusable function - Replace duplicate code with one reusable function to make the code easier to read and repurpose.

Refactor code - Update code so that it is self-documenting and the intent of the code is clear.

Add comments - Adding comments is part of creating self-documenting code. Using comments allows you to leave notes to yourself and/or other programmers to make the purpose of the code clear.

And or and not, like pwsh and dart & or -and

The following examples demonstrate how to use comparison operators with the data types int (integers, whole numbers) and float (number with a decimal point or fractional value). Comparison operators return Boolean results. As you learned previously, Boolean is a data type that can hold only one of two values: True or False.

The comparison operators include:

==   (equality)

!=   (not equal to)

> (greater than)

< (less than)

>= (greater than or equal to)

<= (less than or equal to)

The comparison operators greater than > and less than < can be used to alphabetize words in Python. The letters of the alphabet have numeric codes in Unicode (also known as ASCII values). The uppercase letters A to Z are represented by the Unicode values 65 to 90. The lowercase letters a to z are represented by the Unicode values 97 to 122.

To check if the first letter(s) of a string have a larger Unicode value (meaning the letter is closer to 122 or lowercase z) than the first letter of another string, use the greater than operator: >

To check if the first letter(s) of a string have a smaller Unicode value (meaning the letter is closer to 65 or uppercase A) than the first letter of another string, use the less than operator: <

We also touched on the modulo operator, which is represented by the percent sign: %. This operator performs integer division, but only returns the remainder of this division operation. If we're dividing 5 by 2, the quotient is 2, and the remainder is 1. Two 2s can go into 5, leaving 1 left over. So 5%2 would return 1. Dividing 10 by 5 would give us a quotient of 2 with no remainder, since 5 can go into 10 twice with nothing left over. In this case, 10%5 would return 0, as there is no remainder.

Module 3:
+= is ++ in python

Not quite. First, the variable "multiplier" should be
initialized to equal the number 1. The while loop should run
"while multiplier <= 5:". If the "result" is greater than
25, use the "break" keyword to break out of the loop.
Finally, increment the "multiplier" variable by 1. To review
the materials on this topic, please see the "More While Loop
Examples", "Why Initializing Variables Matters", and
"Infinite Loops and How to Break Them" videos.

In python user for loops when there's a sequence of elements that you want to iterate, use while if you want to complete an action until a condition changes

This code first defines a variable product and initializes it to 1. Then, it uses a for loop to iterate over the range of numbers from 1 to 9. In each iteration of the loop, the variable n is assigned the current number in the range. The code then multiplies product by n. This means that after the first iteration, product will be equal to 1 * 1 = 1. After the second iteration, product will be equal to 1 * 2 = 2. And so on.

The loop will terminate when the value of n is 9. At this point, the value of product will be 1 * 2 * 3 * ... * 9 = 362880.

```
def to_celsius(x):
  return (x-32)*5/9

for x in range(0,101,10):
  print(x, to_celsius(x))
```

The range() function uses a set of indices that point to integer values, which start at the number 0. The numeric values 0, 1, 2, 3, 4 correlate to ordinal index positions 1st, 2nd, 3rd, 4th, 5th. So, when a range call to the 5th index position is made using range(5) the index is pointing to the numeric value of 4.

An optional way to slice an index is by the stride argument, indicated by using a double colon. This allows you to skip over the corresponding number of characters in your index, or if you're using a negative stride, the string prints backwards.

We've covered a bunch of String class methods already, so let's keep building on those and run down some more advanced methods.

The string method lower will return the string with all characters changed to lowercase. The inverse of this is the upper method, which will return the string all in uppercase. Just like with previous methods, we call these on a string using dot notation, like "this is a string".upper(). This would return the string "THIS IS A STRING". This can be super handy when checking user input, since someone might type in all lowercase, all uppercase, or even a mixture of cases.

You can use the strip method to remove surrounding whitespace from a string. Whitespace includes spaces, tabs, and newline characters. You can also use the methods lstrip and rstrip to remove whitespace only from the left or the right side of the string, respectively.

The method count can be used to return the number of times a substring appears in a string. This can be handy for finding out how many characters appear in a string, or counting the number of times a certain word appears in a sentence or paragraph.

If you wanted to check if a string ends with a given substring, you can use the method endswith. This will return True if the substring is found at the end of the string, and False if not.

The isnumeric method can check if a string is composed of only numbers. If the string contains only numbers, this method will return True. We can use this to check if a string contains numbers before passing the string to the int() function to convert it to an integer, avoiding an error. Useful!

We took a look at string concatenation using the plus sign, earlier. We can also use the join method to concatenate strings. This method is called on a string that will be used to join a list of strings. The method takes a list of strings to be joined as a parameter, and returns a new string composed of each of the strings from our list joined using the initial string. For example, " ".join(["This","is","a","sentence"]) would return the string "This is a sentence".

The inverse of the join method is the split method. This allows us to split a string into a list of strings. By default, it splits by any whitespace characters. You can also split by any other characters by passing a parameter.

Lists in Python are defined using square brackets, with the elements stored in the list separated by commas: my_list = ["This", "is", "a", "list"]. You can use the len() function to return the number of elements in a list: len(my_list) would return 4. You can also use the in keyword to check if a list contains a certain element. If the element is present, it will return a True boolean. If the element is not found in the list, it will return False. For example, "This" in my_list would return True in our example. Similar to strings, lists can also use indexing to access specific elements in a list based on their position. You can access the first element in a list by doing my_list[0], which would allow you to access the string "This".

In Python, lists and strings are quite similar. They’re both examples of sequences of data. Sequences have similar properties, like (1) being able to iterate over them using for loops; (2) support indexing; (3) using the len function to find the length of the sequence; (4) using the plus operator + in order to concatenate; and (5) using the in keyword to check if the sequence contains a value. Understanding these concepts allows you to apply them to other sequence types as well.

As we mentioned earlier, strings and lists are both examples of sequences. Strings are sequences of characters, and are immutable. Lists are sequences of elements of any data type, and are mutable. The third sequence type is the tuple. Tuples are like lists, since they can contain elements of any data type. But unlike lists, tuples are immutable. They’re specified using parentheses instead of square brackets.

You might be wondering why tuples are a thing, given how similar they are to lists. Tuples can be useful when we need to ensure that an element is in a certain position and will not change. Since lists are mutable, the order of the elements can be changed on us. Since the order of the elements in a tuple can't be changed, the position of the element in a tuple can have meaning. A good example of this is when a function returns multiple values. In this case, what gets returned is a tuple, with the return values as elements in the tuple. The order of the returned values is important, and a tuple ensures that the order isn’t going to change. Storing the elements of a tuple in separate variables is called unpacking. This allows you to take multiple returned values from a function and store each value in its own variable.

Common sequence operations
Lists and tuples are both sequences and they share a number of sequence operations. The following common sequence operations are used by both lists and tuples:

len(sequence) - Returns the length of the sequence.

for element in sequence - Iterates over each element in the sequence.

if element in sequence - Checks whether the element is part of the sequence.

sequence[x] - Accesses the element at index [x] of the sequence, starting at zero

sequence[x:y] - Accesses a slice starting at index [x], ending at index [y-1]. If [x] is omitted, the index will start at 0 by default. If [y] is omitted, the len(sequence) will set the ending index position by default.

for index, element in enumerate(sequence) - Iterates over both the indices and the elements in the sequence at the same time.

List-specific operations and methods
One major difference between lists and tuples is that lists are mutable (changeable) and tuples are immutable (not changeable). There are a few operations and methods that are specific to changing data within lists:

list[index] = x - Replaces the element at index [n] with x.

list.append(x) - Appends x to the end of the list.

list.insert(index, x) - Inserts x at index position [index].

list.pop(index) - Returns the element at [index] and removes it from the list. If [index] position is not in the list, the last element in the list is returned and removed.

list.remove(x) - Removes the first occurrence of x in the list.

list.sort() - Sorts the items in the list.

list.reverse() - Reverses the order of items of the list.

list.clear() - Deletes all items in the list.

list.copy() - Creates a copy of the list.

list.extend(other_list) - Appends all the elements of other_list at the end of list

map(function, iterable) - Applies a given function to each item of an iterable (such as a list) and returns a map object with the results

zip(*iterables) - Takes in iterables as arguments and returns an iterator that generates tuples, where the i-th tuple contains the i-th element from each of the argument iterables.

Tuple use cases
Remember, there are a number of cases where using a tuple might be more suitable than other data types:

Protecting data: Because tuples are immutable, they can be used in situations where you want to ensure the data you have cannot be changed. This can be particularly helpful when dealing with sensitive or important information that should remain constant throughout the execution of a program.

Hashable keys: Because they're immutable, tuples can be used as keys on dictionaries, which can be useful for complex keys.

Efficiency: Tuples are generally more memory-efficient than lists, making them advantageous when dealing with large datasets.

The tuple() operator

List comprehensions
A list comprehension is an efficient method for creating a new list from a sequence or a range in a single line of code. It is a best practice to add descriptive comments about any list comprehensions used in your code, as their purpose can be difficult to interpret by other coders.

[expression for variable in sequence] - Creates a new list based on the given sequence. Each element in the new list is the result of the given expression.

Example: my_list = [x*2 for x in range(1,11)]

[expression for variable in sequence if condition] - Creates a new list based on a specified sequence. Each element is the result of the given expression; elements are added only if the specified condition is true.

Example: my_list = [x for x in range(1,101) if x % 10 == 0]

Note that tuples do not have comprehensions but a similar functionality can be achieved with:

tuple(i for i in (1, 2, 3))

When to use for loops or list comprehensions
In Python, list comprehensions are generally used for creating new lists from existing ones in a concise and readable manner, especially when the task involves simple transformations or filtering of elements.

for loops are more versatile and are preferred when the operation is more complex, requires multiple lines of code, involves statements other than expression (like print, pass, continue, break), or when you need to iterate over a list without creating a new one.

You want to use dictonaries if you need to search for a certain value.

You can use any mutable data type as the key

In Python, methods are behaviors associated with object parameters that modify the state of that object. They are essentially functions that belong to a specific instance of a class. This means that calling a method on a list, for example, only modifies that instance of the list, and not all lists globally.

Methods in Python fall into several categories:

Instance methods

Class methods

Static methods

Intro to Python Final Project:

Daily Report to track machine use
Generate report of login to machines at the time

Attributes:
Date, User, Machine, Type

Care ab:
Login, Logout, they are strings.

Output:

List all machine names, list users logged in. Print it on the screen.

Machine name by colon then users in same line, seperated by commas.

Mod 2:

No notes from mod1 in mod2

File descriptor:

A token, generated by the os, that allows programs to do more operations with the file
Stored as an attribute of the file object in python

Use with so that you don't cause a race condition when opening files

File modes:
R : read only
W : write only
A : append content
R+: read/write

IF YOU OPEN A FILE FOR WRITING AND THE FILE ALREADY EXISTS, THE OLD CONTENTS WILL BE DELETED AS SOON AS THE FILE IS OPENED!!!!!!!

U can use / in python for windows, it is perfered since \ is a special char and you'd have to use:
C:\\dir\\subdir

Os.getcwd()

Os module is an abstraction layer

Os.path.getsize is in bytes

The OS module
Python's OS module, or the miscellaneous operating system interface, is very useful for file operations, directories, and permissions. Let's take a look at each.

File operations
File names can be thought of as two names separated by a dot. For example, helloworld.txt is the file name and the extension defines the file type. OS provides functions to create, read, update, and delete files. Some of the basic functions include:

Opening and closing files

Reading from and writing to files

Appending to files

Directories
OS also provides functions to create, read, update, and delete directories, as well as change directories and list files. Knowing how to use these functions is key to working with files. For example, os.listdir(path) returns a list of all files and subdirectories in a directory.

Permissions
Having the ability to update file permissions is an important aspect of making installations from a terminal window. The os.chmod() provides the ability to create, read, and update permissions for individuals or groups.

Things to keep in mind
One thing to be aware of is that Python treats text and binary files differently. Because Python is cross-platform, it tries to automatically handle different ASCII line endings. If you're processing a binary file,

make sure to open it in binary mode so Python doesn't try to "fix" newlines in a binary file.

A best practice is to always close() a file when you're done reading or writing to it. Even though Python usually closes them for you, it's a good signal to other people reading your code that you're done with that file. Make sure to catch any potential errors from filesystem calls, such as permission denied, file not found, and so on. Generally, you wrap them in try/except to handle those errors.

Key takeaways
There are several ways to manage files and directories in Python. One way is to use low-level functions in the OS and SYS modules that closely mimic standard Linux commands. Another way is to utilize the Pathlib module, which provides an object-oriented interface to working with the file systems.

`` Means code blcok

Regex

More powerful then find()

Ex grep:

`grep thon /usr/`

Grep Rules:

A dot matches any character - Wildcard - Like PowerShell's `*`
^ - Circumflex - Beginning of the line
$ - End of the line
.* matches any char between params

Always use raw strings

\w matches any alphanumeric characters - no space
\d - digits
\s - whitespace
\b - word boundaries

{} repetition qualifier
Advanced regular expressions—commonly referred to as advanced regexes—are used by developers to execute complicated pattern matching against strings. In this reading, you will learn about some of the common examples of advanced regular expressions.

Alterations
An alteration matches any one of the alternatives separated by the pipe | symbol. Let's look at an example:

r"location.*(London|Berlin|Madrid)"

This line of code will match the text string location is London, location is Berlin, or location is Madrid.

Matching only at the beginning or end
If you use the circumflex symbol (also known as a caret symbol) ^ as the first character of your regex, it will match only if the pattern occurs at the start of the string. Alternatively, if you use the dollar sign symbol $ at the end of a regex, it will match only if the pattern occurs at the end. Let's look at an example:

r"^My name is (\w+)"

This line of code will match My name is Asha but not Hello. My name is Asha.

Character ranges
Character ranges can be used to match a single character against a set of possibilities. Let's look at a couple of examples:

r"[A-Z] This will match a single uppercase letter.

r"[0-9$-,.] This will match any of the digits zero through nine, or the dollar sign, hyphen, comma, or period.

The two examples above are often combined with the repetition qualifiers. Let’s look at one more example:

r”([0-9]{3}-[0-9]{3}-[0-9]{4})”

This line of code will match a U.S. phone number such as 888-123-7612.

Backreferences
A backreference is used when using re.sub() to substitute the value of a capture group into the output. Let’s look at an example:

>>> re.sub(r”([A-Z])\.\s+(\w+)”, r”Ms. \2”, “A. Weber and B. Bellmas have joined the team.”)

This line of code will produce Ms. Weber and Ms. Bellmas have joined the team.

Lookahead
A lookahead matches a pattern only if it’s followed by another pattern. Let’s look at an example:

If the regex was r”(Test\d)-(?=Passed)” and the string was “Test1-Passed, Test2-Passed, Test3-Failed, Test4-Passed, Test5-Failed” the output would be:

Test1, Test2, Test4

Data streams:

Exit status - value return to a shell

0 is proceeds
Anything else is faliure

Echo $? To get exit codes

Subprocesses:

Bash: The most commonly used shell on Linux

Command line arguments: Inputs provided to a program when running it from the command line

Environment variables: Settings and data stored outside a program that can be accessed by it to alter how the program behaves in a particular environment

Input / Output (I/O): These streams are the basic mechanism for performing input and output operations in your programs

Log files: Log files are records or text files that store a history of events, actions, or errors generated by a computer system, software, or application for diagnostic, troubleshooting, or auditing purposes

Standard input stream commonly (STDIN): A channel between a program and a source of input

Standard output stream (STDOUT): A pathway between a program and a target of output, like a display

Standard error (STDERR): This displays output like standard out, but is used specifically as a channel to show error messages and diagnostics from the program

Shell: The application that reads and executes all commands

Subprocesses: A process to call and run other applications from within Python, including other Python scripts

Tests:

Test cases

Unit tests, used to verify that small, isolated parts of a program are correct

Edge case, inputs to our code that produce unexpected results, and are found at the extereme ends of the ranges of input we imagine our programs will typically work with

Ticky service

Mod 3

Github:

for you to revert or undo your mistakes. Take a look at the helpful commands below.

git checkout
 is used to switch branches. For example, you might want to pull from your main branch. In this case, you would use the command git checkout main. This will switch to your main branch, allowing you to pull. Then you could switch to another branch by using the command git checkout <branch>.

git reset
  can be somewhat difficult to understand. Say you have just used the command git add. to stage all of your changes, but then you decide that you are not ready to stage those files. You could use the command git reset to undo the staging of your files.

There are some other useful articles online, which discuss more aggressive approaches to
 resetting the repo
 (Git repository). As discussed in this article, doing a hard reset can be extremely dangerous. With a hard reset, you run the risk of losing your local changes. There are safer ways to achieve the same effect. For example, you could run git stash, which will temporarily shelve or stash your current changes. This way, your current changes are kept safe, and you can come back to them if needed.

git commit --amend
 is used to make changes to your most recent commit after-the-fact, which can be useful for making notes about or adding files to your most recent commit. Be aware that this git --amend command rewrites and replaces your previous commit, so it is best not to use this command on a published commit.

git revert
 makes a new commit which effectively rolls back a previous commit. Unlike the git reset command which rewrites your commit history, the git revert command creates a new commit which undoes the changes in a specific commit. Therefore, a revert command is generally safer than a reset command.

For more information on these and other methods to undo something in Git, checkout this
Git Basics - Undoing Things
 article.

Additionally, there are some interesting considerations about how git object data is stored, such as the usage of SHA-1.

SHA-1 is what's known as a hash function, a cryptographic function that generates a digital fingerprint of a file. Theoretically, it's impossible for two different files to have the same SHA-1 hash, which means that SHA-1 can be used for two things:

Confirming that the contents of a file have not changed (digital signature).

Serving as an identifier for the file itself (a token or fingerprint).

Git calculates a hash for every commit. Those hashes are displayed by commands like git log or in various pages on Github. For commands like git revert, you can then use the hash to refer to a specific commit.

Feel free to read more here:

SHA-1 collision detection on GitHub.com

Even the most accomplished developers make mistakes in Git. It happens to everyone, so don't stress about it. You have these and other methods to help you revert or undo your mistakes.

git branch

```
$ git branch
```
can be used to list, create, or delete branches.

git branch <name>

```
$ git branch <name>
```
can be used to create a new branch in your repository.

git branch -d <name>

```
$ git branch -d <name>
```
can be used to delete a branch from your repository.

git branch -D <name>

```
$ git branch -D <branch>
```
forces a branch to be deleted.

git checkout <branch>

```
$ git checkout <branch>
```
switches your current working branch.

git checkout -b <new-branch>

```
$ git checkout -b <new-branch>
```
creates a new branch and makes it your current working branch.

git merge <branch>

```
$ git merge <branch>
```
joins changes from one branch into another branch.

git merge --abort

```
$ git merge --abort
```
can only be used after merge conflicts. This command will abort the merge and try to go back to the pre-merge state.

git log --graph

```
$ git log --graph
```
prints an ASCII graph of the commit and merge history.

git log --oneline

$ git log --oneline
 prints each commit on a single line.

Terms and definitions from Course 3, Module 2
Branch: A pointer to a particular commit, representing an independent line of development in a project

Commit ID: An identifier next to the word commit in the log

Fast-forward merge: A merge when all the commits in the checked out branch are also in the branch that's being merged

Head: This points to the top of the branch that is being used

Master: The default branch that Git creates for when a new repository initialized, commonly used to place the approved pieces of a project

Merge conflict: This occurs when the changes are made on the same part of the same file, and Git won't know how to merge those changes

Rollback: The act of reverting changes made to software to a previous state

Three-way merge: A merge when the snapshots at the two branch tips with the most recent common ancestor, the commit before the divergence

git remote

$ git remote
 allows you to manage the set of repositories or "remotes" whose branches you track.

git remote -v

$ git remote -v
 is similar to $ git remote, but adding the -v shows more information such as the remote URL.

git remote show <name>

$ git remote show <name>
 shows some information about a single remote repo.

git remote update

$ git remote update
 fetches updates for remotes or remote groups.

git fetch

$ git fetch
 can download objects and refs from a single repo, a single URL, or from several repositories at once.

git branch -r

$ git branch -r
 lists remote branches and can be combined with other branch arguments to manage remote branches.

Terms and definitions from Course 3, Module 3
Application Programming Interface (API) key: This is an authentication token that calls an API, which is then called to identify the person, programmer, or program trying to access a website

Computer protocols: Guidelines published as open standards so that any given protocol can be implemented in various products

Distributed: Each developer has a copy of the whole repository on their local machine

GitHub: A web-based Git repository hosting service, allowing users to share and access repositories on the web and copy or clone them to a local computer

Merge: An operation that merges the origin/master branch into a local master branch

Private key: A secret and secure cryptographic key that must be kept confidential and protected and is used to decrypt data that has been encrypted with the corresponding public key

Public key: A safety cryptographic structure frequently employed to establish secure communication through data encryption or to validate the authenticity of a digital signature

Rebasing: The base commit that's used for a branch is changed

Remote branches: Git uses read-only branches to keep copies of the data that's stored in the remote repository

Remote repositories: Repositories that allow developers to contribute to a project from their own workstations making changes to local copies of the project independently of one another

Secure Shell (SSH): A robust protocol for connecting to servers remotely

SSH client: This establishes a connection to the SSH server, ensuring a secure interaction, where the client makes access requests

SSH key: An access credential

SSH protocol: Standard commonly used for logging in to servers remotely on the principle of public-key encryption

SSH server: This establishes secure network connections, undergoes mutual authentication, and initiates encrypted login sessions or file transfers

Besides serving as a repository for your code and tracking changes to your code over time, GitHub also includes tools to help manage your software project.

GitHub Projects
GitHub offers multiple tools to manage and plan your work. For example, GitHub Projects is a flexible tool for tracking and managing work on GitHub. You can use Projects to create an adaptable spreadsheet, task-board, and road map which integrates with your issues and pull requests. With GitHub projects, you can filter, sort, and group your issues and pull requests and customize to fit your team's workflow. Projects can be created in a repository, and then issues can be added to them.

GitHub Project task board with categories: incoming, needs review, by code owner, and review in progress.
GitHub Issues
GitHub Issues is a part of GitHub Projects, and it provides a way to track tasks that you need to complete. An issue can be a bug, a feature request, or a housekeeping task (like upgrading a library to the latest version). Issues can have extensive text and descriptions attached to them, including screenshots and snippets of code. Issues can be discussed, commented on, assigned to people, and tagged.

Here's a screenshot of the top open issues for a very active Python application on GitHub:

GitHub Issues example with a list of issues for a specific project.
As issues are worked, and pull requests linked to those issues are completed, Issues will automatically move into the next column, and then the next, until they're closed. You can also drag issues to another column, and GitHub will update the status (and possibly assignee and other fields) to match.

Traditional project management
You can also view open issues in a more traditional project management format, with status, assignees, estimates, and more:

Traditional project management format of a project called OctoArcade Invaders.
GitHub also supports extensive automation. You can define workflows that update issues or projects automatically as issues change or comments are added.

Resources for more information
A Quick Guide to Using GitHub for Project Management

This article provides a brief overview of project management tools on GitHub.

GitHub for project management

This lesson offers detailed descriptions of GitHub's project management tools.

Using GitHub as your Project Management Tool

This video provides examples on GitHub project management tools.

GitHub Issues: Project Planning for Developers

This GitHub page shows the many project management tools available for developers.

IT skills in action
Well done! You've gained a strong foundation in various crucial aspects of IT, from version control to project management. You've also practiced how to:

Implement version control using git and Github

Branch and merge your work

Secure and restore repositories

Resolve code conflicts

Run code reviews and manage pull requests

Use versioning to track and manage projects

Now, let's put your skills into action with a real-world scenario that encompasses the lessons you've learned.

The project
Imagine you're part of an IT team responsible for developing and managing a software project. Your team is using Git for version control, collaborating on coding tasks, and ensuring project success. Let's walk through the process step by step.

Project steps
Before Version Control: Before diving into code, ensure your team is aligned on the project's scope, goals, and responsibilities.

Version control systems: Choose Github as your version control system to track changes, collaborate effectively, and maintain a history of your project.

Using git: Start by initializing a Github repository, committing your initial code, and using git status and git log to manage and track changes.

Advanced git interaction: Use advanced commands like git diff to visualize changes, git stash to temporarily hide changes, and git tag to mark significant milestones.

Undoing things: Use git reset and git revert to undo changes and address errors in a controlled manner.

Branching and merging: Create branches for feature development using git branch, switch between branches with git checkout, and merge changes using git merge.

Secure shells & API keys: Ensure security by using SSH keys and managing sensitive data like API keys properly.

Solving conflicts: Resolve conflicts that arise from merging branches using git merge or pull requests.

Pull requests: Open pull requests to propose changes, review code, and discuss modifications with your team.

Code reviews: Participate in code reviews to maintain code quality, identify improvements, and ensure best practices.

Managing projects: Organize your project using project boards, milestones, and issues to track progress and prioritize tasks.

Putting it all together
Imagine you're assigned to add a new feature to your project: a user authentication system. Here's how you'd apply your skills:

Before version control: Working with your development team and stakeholders you define the feature's scope and priorities. From the business requirements you develop user stories from which the team can build out tasks. Review the tasks your team created and discuss expected outcomes.

Version control systems: You create a feature branch for the authentication system on the app's existing repository that is already located on github. Your team uses this new branch to begin to work on the tasks associated with the feature request.All progress is tracked in real time and documented with comments in Github.

```
# Create a new feature branch

git checkout -b feature/user-authentication
```

Advanced git interaction: You use git diff to view and compare code changes and look back at the history of changes. When needed you can use git diff to compare whole branches as the feature becomes more robust. As you get closer to completing the feature you create tags to mark development milestones. When feature release is approaching, you can use a milestone to share progress with stakeholders.

```
# View code changes

git diff

# View commit history

git log

# Create a new tag

git tag v1.0.0

# Compare branches

git diff feature/user-authentication main
```

Undoing things: As you encounter issues, you have stable milestones you know you can restore back. You can stash away pending changes or, safely undo changes using Git's commands.

```
# Stash changes

git stash

# Restore changes from stash

git stash pop

# Undo changes in working directory

git checkout -- <file>
```

Branching and merging: Your team makes sure to keep up with branching and merging changes. The team tests their changes in the feature branch to avoid introducing any issues or bugs into the main branch.

```
# Merge changes from feature branch to main

git checkout main

git merge feature/user-authentication

# Delete feature branch

git branch -d feature/user-authentication
```

Solving Conflicts: As code conflicts arise during merging, you attempt to automerge. When deeper conflicts arise, you gather your team and address them collaboratively.

```
# Attempt to automerge

git merge feature/user-authentication

# Resolve conflicts manually

# Edit files to resolve conflicts

git add <resolved-files>

git commit -m "Resolved conflicts"
```

Pull requests and code reviews: One of your team members opens up a pull request for your feature branch. It is finally time to merge our feature into the main branch. Automated tests run against the code in question and your team schedules a code review. You prepare to gather and track feedback.

```
# Push changes and open pull request

git push origin feature/user-authentication

# Automated tests run in CI/CD pipeline

# Pull request is reviewed

# Feedback is addressed
```

Code reviews: All concerned parties participate in code reviews. Team members address the group and review their code additions. Tests and metrics are also reviewed. The team collaborates at addressing feedback and ensuring high-quality code.

Managing projects: Throughout the project, and even after development efforts have concluded, you continue to track the progress of your feature using project boards, milestones, and issues. Development is iterative and your team will continue to work on features as feedback and requests come in from stakeholders.

By applying your skills across the development life-cycle, you've successfully contributed to the project's growth and demonstrated your expertise in IT and project management.

Key takeaways
Throughout this guided activity, you've delved into the practical application of various IT skills, following a step-by-step process that encapsulates the skills you’ve learned. You've navigated the world of version control systems, using Git's essential functionalities, branching strategies, and remote repository interactions. With a keen eye for detail, you've tackled code reviews and confidently resolved conflicts, ensuring the seamless collaboration essential for effective software development. Your journey also encompassed essential project management aspects, where you employed project boards, milestones, and issue tracking to oversee and guide your projects' evolution. Feel confident to harness your IT skills in real-world contexts, paving the way for efficient, collaborative, and successful software development endeavors.

Tailor your resume
As you prepare for your job search, you will need to create or update your resume to reflect your experience in order to apply for roles like:

Automation Engineer

Entry-level Python Developer

IT Support Specialist II

Entry-level Software Engineer

Network Engineer

…and other similar job titles

You have learned so much during this certificate program, and it is important that your resume reflects that. An effective resume highlights your skills and experience and is tailored to the position you are applying for. Let's explore how to make your resume stand out by incorporating your new Python automation skills and your previous experience.

Tailor the content
Identify what is important to the potential employer. What does the employer want to know about you? Make sure that you carefully read the job description and notice which skills are mentioned. You can also read several job descriptions for the same type of role to identify which skills and requirements show up frequently. For instance, although specifics will vary by role and employer, many Python automation-related roles require the ability to effectively organize and coordinate across teams and projects, manage multiple tasks simultaneously, and communicate effectively. You should take note of these skills and be sure to highlight them using similar terms on your resume.

Create one primary resume to edit and tailor to each job application. You should make sure that the order of your skills and qualifications matches the job description. In doing this, you are making sure that the things that are most important to the employer are at the top.

Match the language used in the job description. Some employers use automation software to filter resumes. If the job description uses keywords like cloud services and risk management, make sure your resume uses those keywords, too.

Use Python automation terminology. This will help the hiring manager reading your resume understand how your past experience is relevant to the role for which you are applying.

Decide what not to include on your resume. You may have some skills that are important to you, but those same skills may confuse or distract the hiring managers reading your resume.

Highlight how your experience and skills are relevant to the job. If you have been working as an IT Support Specialist but want to become a Python Automation Engineer, your troubleshooting skills will be essential in your new role. Make sure to point out how those skills will be beneficial to the employer.

Choose an appropriate format
No matter what layout or template you choose for your resume, there are several things you should keep in mind

The design of your resume should be simple and easy to understand for both human and artificial intelligence readers. You don't want your resume to be discarded before a real person has a chance to read it!

Your resume should be easy to read and communicate all of the important information in short bullet points.

Your resume should be one- to two-pages long and contain only the last ten to fifteen years of relevant experience. It is appropriate to use two columns on a one-page resume, but if your resume is two pages, be sure to use the entire width of the page.

Update the relevant sections
Once you have determined the appropriate format for your resume, you will need to update each of your resume's major sections, which include:

Contact information

Professional summary

Core competencies

Professional experience

Education and certifications

Pro tip: Resumes should be written in the third person and should not contain personal pronouns.

Let’s discuss how to incorporate your new skills into these sections of your resume.

Contact information
Your header should contain your contact information and should go at the top of your resume.

Your header should include the following information:

Your name in a larger font than the rest of your resume

The city and state you live in (you do not need to include your street address for privacy purposes)

Your phone number and a link to your email address

Link to your LinkedIn profile URL

Links to any other personal websites or portfolios, if applicable to the role you are applying for

Your header should be relevant, simple, and easy to read. Here is an example of a resume header:

The left side of the headers shows the name Samara Mittal and the right side shows this person's location and contact info.
Professional summary
Below your header, include a professional summary.

Use your summary to set the tone. Your summary should be one to three lines and should clearly state why you are the best candidate for the position. It should showcase the most important things you want the reader to know about you. If you are applying for a new role, you will want to update your industry specialty. You likely have experience that can be related to critical thinking and complex problem solving. You will want to incorporate that relevant experience into your new professional summary. Make sure you tailor your description of yourself to the role you are applying for.

Merge the description of the role you are applying for with your experience. Here is an example:

Automation Engineer with two years of demonstrated success in complex problem solving. Skilled in cross-functional collaboration and project execution. Articulate communicator who thrives in a results-driven collaborative environment.

Use keywords from the job description to describe yourself. If the job description states that the company is looking for a candidate with knowledge of cloud computing, Linux, or Bash scripting, you should add that to your resume—you have gained that knowledge with this certification.

Once you have your professional introduction, your next sentence should describe how your unique expertise will make you valuable to the employer.

Pro tip: Don’t forget to use this section to highlight something that makes you stand out from other applicants. Use an accomplishment from a previous role to show the employer what you can offer them. Take a look at this example of a professional summary section:

Professional Summary heading the the summary beneath it.
Now that you have your heading and professional summary updates, let’s move on to the core competencies section of your resume.

Core competencies
Your core competencies should be a bulleted list of the most relevant skills applicable to the position you are applying for.

Pro tip: Scan the job description for core competencies you have gained during this certification and your past experience then use those skills as bullet points in this section. Make sure to keep this section relatively short, with four to eight bullets. Here is an example of a Python Automation Engineer resume core competencies section:

Core competencies header with a bullet point list of skills.
Now that you have showcased who you are and what makes you the best candidate for the job, it is time to tell the story of what you have accomplished throughout your career in the professional experience section.

Professional experience
The professional experience section of your resume provides a summary of the roles and positions you have held in your career. List at least three positions in reverse chronological order and only include what is most relevant to the position you are applying for.

Your professional experience will not change much from previous resumes, because you can't change the past roles you have held. However, you can possibly rewrite some of your bullets to relate them to your target job's requirements. Make sure you are tying the industry lingo back to your previous experience to show the reader—usually a hiring manager—how your skills relate to the advertised position. You may be able to use terms like troubleshooting, critical thinking, testing, implementation, and software maintenance to show the reader that your past experience translates to a Python Automation or Software Engineering role, for example.

Pro tip: Make sure your resume conveys how your past accomplishments are valuable to the role you are applying for. Show the reader how you can make a difference in their organization. An easy way to remember this is through the P.A.R.I.S. framework:

Problem that needed to be solved

Action(s) I took

Result of action(s)

Impact on project (users, quality, etc.)

Supporting evidence (awards, bonus, etc.)

Below is an example of a professional experience section from a Python Automation Engineer's resume:

Job titles, locations, and time periods with descriptions.

Education and certificates
Now that the majority of your resume has been updated with your new skills and knowledge, it is time to update your Education and Certifications section. In this section of your resume, you should include any degrees beyond your high school diploma in reverse chronological order. For each degree, list the degree you earned, institution, location, and date of graduation. This section should also list any professional certifications, licenses, or credentials you hold. It is here where you will list your new Google Professional Certificate. Here is an example of an education and credentials section of a Python Automation Engineer resume:

Education header with certifications and degrees acquired.
Your resume is now updated and ready to use for your target job! You have revised your professional summary, added newly-acquired core competencies, related past professional experience to your target position, and added this certification to your resume.

Pro tip: It is always a good idea to have someone review your resume for any spelling or grammatical errors. Recruiters and hiring managers often toss resumes aside that contain typos. Once you are sure your resume is error-free, it is time to start your job search!

Writing a Cover Letter
A cover letter is a personal introduction to promote yourself. It serves as a companion document to a resume. Its main purpose is to elaborate on your professional skills, motivations, and why you should be viewed as the best candidate for a job.

There are three types of cover letters:

Networking—Addressed to individuals to ask for their help in finding a job at their company

Prospecting—Addressed to companies to explore all open job opportunities

Application—Addressed to hiring managers to emphasize your fit for a specific job

This reading focuses on helping you write application cover letters. Employers have “who, what, where, when, why, and how” types of questions when they gather information about job applicants.

Your resume answers:

what you have done

where you worked

when you were employed.

Your application cover letter describes in more detail:

who you are

why you want the job

how you will be successful in the role

Preparing to write a cover letter
Pro tip: Not all job applications require a cover letter. When a cover letter is stated as being optional, it’s best to consider how much a cover letter might improve your standing. The following are common situations when people prefer to include a cover letter:

When starting out in a career (early career applicant)

When making a career transition

When experience, education, or training isn’t an exact match with the listed requirements

When entering a crowded field of applicants

Many people agree that no cover letter is better than a poorly written one! Even if a cover letter has no obvious errors, submitting a few paragraphs with very general statements isn’t going to help you that much. Before you write a cover letter, follow these steps to ensure you create a meaningful one.

Step 1: Research the company or organization
You can find out a lot about a company from these methods:

Browse the company’s website

Follow the company on social media including LinkedIn

Perform a search on the company’s financial standing and investors, if applicable

Perform a search on the company’s known competitors

Ask your relatives, friends, and colleagues what they know about the company

Step 2: Inventory the required skills from the job description
Read the job description carefully and determine what you think are the most important skills for an applicant to have and why.

Step 3: Prioritize your matching skills from strongest to weakest
Based on the skills you identified in the previous step, identify your skills that match or are most closely aligned (associated) with them. Next, prioritize your matching skills from the strongest to the weakest. People often skip this ordering process. Ranking your skills enables you to emphasize your strongest skills first in your cover letter.

Parts of a cover letter

Pro tip: A cover letter is between 250-400 words in length and doesn't exceed one page.

Review the goals for each section of a cover letter below.

Introduction
The primary goals of the introduction section of a cover letter are to:

Identify the position you're applying for

Show your enthusiasm for the company

Encourage people on the hiring team to learn more about you

Example I'm applying for the Python Developer position, and can't imagine a more exciting role. As a frequent and avid user of your services, I'm eager to pursue this career opportunity.

Body Section
The goal of the body section of a cover letter is to describe how your skills apply to the open position. Suppose the job description has these qualifications:

Experience with Python scripting

At least 1 year of IT troubleshooting experience

Critical thinking skills with ability to solve complex problems

The following example shows how you can map your experiences to these qualifications in the body of your cover letter.

Example

I completed the Google ____ Professional certificate and have previous customer service experience in retail electronics. I was responsible for helping customers choose the right devices for their needs, and solve any problems they had with those devices. I also helped keep the onsite computers healthy by identifying and solving any technical problems. With my attention to detail, I can help your IT team solve problems quickly and efficiently.

Pro tip: Avoid the temptation to rehash the content of your resume. A warning sign is if your cover letter has essentially the same information as your resume but in a paragraph format.

Pro tip: Focus on what you can do for the company rather than on how you would benefit from being hired for that position. The difference between being company-focused and self-focused can be subtle, as in the following sentences:

I would like to develop automation solutions to keep your company's IT structure solid and provide efficiency to the company's IT systems (company-focused; what you will do for the company).

I would like to grow my Python skills by developing cloud automation solutions for efficient virtual machine management (self-focused; what you would like to gain by being in the role).

Closing
The goal of the closing section of a cover letter is to restate your interest in the company and position. It is also used to indicate your expectations, such as scheduling an interview, being considered for other jobs, or a timeframe for follow up.

Example

Thank you for taking the time to review my resume for this position. I'm confident I can excel in this role using my combined work experience and skills from the Google ____ Professional Certificate. I'm looking forward to an interview and request the privilege to follow up on my application's progress in the coming weeks.

Proofread your cover letter
Many errors in cover letters are caused by copying and pasting text from one cover letter to another. After you write your cover letter, proofread it carefully to catch these common things:

Awkward formality—Few people call people Sir or Madam nowadays, so you shouldn't use these in your cover letter either. Also refrain from using "To whom it may concern" which sounds highly impersonal. If you don't know the name of the hiring manager, use "Dear Hiring Team." Likewise, consider using "Best regards" instead of "Sincerely" which sounds a little outdated.

Misspelled words (especially those that sound the same but are spelled differently). For example, "affect" and "effect," "then" and "than," and "your" and "you're."

Mismatched skills—Make sure you aren't incorporating skills for the wrong job description in your cover letter. This happens with copying and pasting.

Passive voice—Use active voice whenever possible: "I revised the ads" instead of passive voice: "I ensured that the ads were revised."

Long anecdotes—Save stories that describe any past results you achieved for when you are being interviewed.

Key takeaways
Cover letters help introduce the best points about yourself to a potential employer. Make sure that your cover letter doesn't simply rehash the skills outlined in your resume, but adds value by describing how your skills align with the job requirements and how you would be successful in the role. To write the best cover letters, it's helpful to research the company, identify the most important skills from the job descriptions, and prioritize and include your matching and relevant skills.

Mod 4

Troubleshooting and Debuging:

Steps to solve problems:

1. Gather information
2. Find root cause
3. Perform necessary remidation

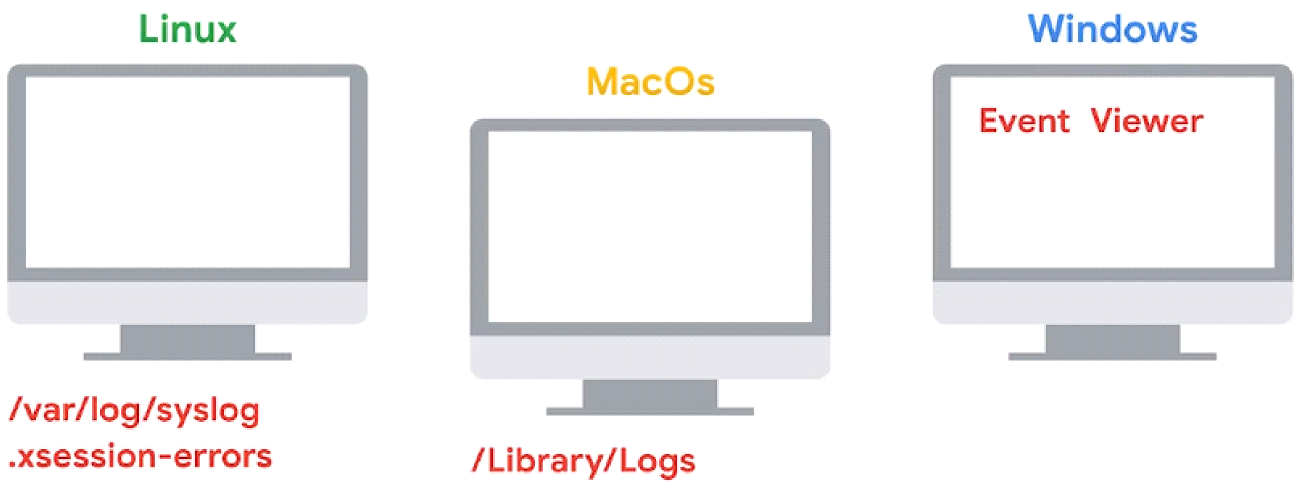
Strace - find issues with program, find system calls.
Ltrace - find issues with library calls
Top - cpu info

?s to ask:

What were you trying to do? What steps did you follow? What was the expected result? What was the actual result?

Logs to look at with issues:

Linux
MacOs
Windows
Event Viewer



Strace and ltrace as always

One possible culprit could be too much disk input and output. To get more info on this, we could use iotop, which is a tool similar to top that lets us see which processes are using the most input and output. Other related tools are iostat and vmstat, these tools show statistics on the input/output operations and the virtual memory operations. If the issue is that the process generates too much input or output, we could use a command like ionice to make our backup system reduce its priority to access the disk and let the web services use it too. What if the input and output is not the issue? Another option would be that the service is using too much network because it's transmitting the data to be backed up to a central server and that transmission blocks everything else. We can check this using iftop, yet another tool similar to top that shows the current traffic on the network interfaces.

If the backup is eating all the network bandwidth, we could look at the documentation for the backup software and check if it already includes an option to limit the bandwidth. The rsync command, which is often used for backing up data, includes a
-bwlimit, just for this purpose. If that option isn't available, we can use a program like Trickle to limit the bandwidth being used. But what if the network isn't the issue either? Remember, we need to put our debugging creativity to work, and come up with other possible reasons for why it's failing.

Another option could be that the compression algorithms selected is too aggressive, and compressing the backups is using all of the server's processing power. We could solve this by reducing the compression level or using the nice command to reduce the priority of the process and accessing the CPU. If that's still not the case, we need to keep looking, check the logs to see if we find anything that we missed before. Maybe look online for other people dealing with similar problems related to interactions of the backing up software with the web surfing software, and keep doing this until we come up with something that could be causing our problem. I know this sounds like a lot of work, but it's usually not that bad. In general, by using the tools available to us, we can find enough info to land on the right hypothesis after only a few tries and with experience, we'll get better at picking up the most likely hypothesis the first time around. Up next, we'll talk about a tricky type of technical problem that we all have to face, intermittent issues.

Linear search:

One possible approach would be to start from the first entry and then check if the name is the one that we're looking for.
If it doesn't match, move to the second element and check again, and keep going until we find the employee with the name we're looking for, or we get to the end of the list. This is called a linear search.

WC - counts words and lines in a file
Wc -l for lines

Terms and definitions from Course 4, Module 1
Binary search: A search algorithm used to find a specific item in a sorted list or array by repeatedly dividing the search space in half until the desired item is found or determined to be absent

Bisecting: Dividing in two, also a Git command

Debuggers: Tools that follow the code line by line, inspect changes in variable assignments, interrupt the program when a specific condition is met, and more

Debugging: The process of identifying, analyzing, and removing bugs in the actual code of a system in the application

Linear search: The process of searching each line of data until the desired data entry is located

Observer effect: The idea that observing a phenomenon alters the phenomenon

System calls: The calls that the programs running on our computer make to the running kernel

Troubleshooting: The process of solving any kind of problem in the system running the application

Ab -n 500 site.example.com/

Used to check if a website is working as expected. Apache benchmark

Nice to start a processs with a different priority
Renice to start a process that is running with a different priority
Daemonize - runs program as if it was a daemon

There are robust tools at your disposal for finding and diagnosing performance bottlenecks in computer systems. This guarantees a seamless and refined operational experience. Windows, Linux, and macOS all offer a wide range of methodologies and tools for monitoring and fine-tuning system performance.

Windows processes
Windows Process Monitor, also known as Sysinternals, is a powerful monitoring tool that serves as an advanced task manager. It provides real-time insight into various aspects of the system, including file system operations, registry changes, processes, and threads. The tool excels at diagnosing file access issues, analyzing system configurations, and understanding processes.

You can use Process Monitor to track down bugs, detect unauthorized changes to the Registry, and investigate system crashes, making it an indispensable tool for system troubleshooting. With detailed event properties and a wide range of filtering options, you can pinpoint root causes more efficiently by focusing on specific processes.

When combined with logging, reporting, and monitoring tools, Process Monitor can enhance the effectiveness of diagnosing and resolving complex issues. It can also be useful for detecting suspicious applications running in the background unnoticed.

To fully explore Process Monitor's capabilities, read more about it
here
.

Linux performance
To enhance your Linux system's performance, you can use specialized tools that offer real-time insights into CPU, memory, disk I/O, and network activity for quick performance bottleneck detection. Some of these tools include Perf-tools, bcc/BPF, and bpftrace.

To further optimize your system, use a static analysis tool to examine your code and configurations for potential improvements. The use of benchmarking tools can also be helpful for assessing your system's performance under different workloads and revealing areas that may need to be improved.

Customizing your Linux system using tuning utilities is a powerful strategy to tailor settings and achieve a faster and more responsive setup. For instance, the SAR (System Activity Reporter) is especially useful for analyzing performance trends and identifying recurring issues over time.

You can effectively troubleshoot problems, fine-tune performance, and ensure the smooth and efficient operation of your Linux system by incorporating these tools along with historical data.

To gain real-time insights into your Linux system's performance, read more
here
.

The USE method
The USE Method is essential for optimizing system performance and troubleshooting servers. It helps identify resource bottlen ecks and performance issues by analyzing Utilization, Saturation, and Errors. Resources like CPUs, memory, storage, and network interf aces can be measured for busy time, additional workload capacity, and errors.

To pinpoint problems and relationships, the USE Method suggests creating a resource list and a Functional Block Diagram. This helps avoid data overload and provides a clear visualization of the system's components and their interactions.

This method is adaptable to cloud computing environments to assess how software resource controls impact performance. This methodology provides a simple and effective approach to optimizing system performance.

For more detailed information, including specific checklists for different operating systems and guidance, read more
here
.

macOS Activity Monitor
Activity Monitor in macOS allows you to monitor and manage system performance easily. You can optimize Mac performance with Activity Monitor's insights into process activity, resource usage, and energy consumption. Activity Monitor identifies unresp onsive apps or processes, monitors energy usage, tracks overall energy impact, and displays real-time system status. It enables you to troubleshoot issues and optimize battery life, ensuring smooth and responsive operation.

For detailed instructions and information on how to use this utility, refer to Apple's Activity Monitor User Guide
here
.

Windows Performance Monitor
Performance Monitor is a versatile and customizable tool that analyzes your system's performance. By identifying and resolvin g hardware problems, poorly designed apps, excessive resource usage, or malware, it ensures smooth and efficient operation. Hav ing real-time data on memory, network, disks, and processors lets you monitor key components and quickly resolve problems. You can configure counters, set data collectors, and analyze reports to optimize your system.

For more information on maintaining optimal system performance, read more
here
.

Windows Resource Monitor
To get real-time insights into your computer's resource usage in Windows, use the Resource Monitor (resmon.exe) tool. It helps identify causes of slowdowns like hardware issues, poorly designed apps, and malware. Access it by searching for "resmon" or "Resource Monitor." Navigate between Memory, Disk, and Network sections for deeper analysis. Be cautious with CPU processes to avoid sy stem instability.

Resource Monitor helps you understand your system's resource usage. To learn more about the Resource Monitor, read more
here
.

Windows Process Explorer
The Process Explorer v17.05 software is primarily used for file monitoring and analyzing processes on Windows computers. It p rovides detailed information about active processes, handles, and DLLs. Processes and their accounts are displayed in the top window, while handles and DLLs are displayed in the bottom window. In addition to troubleshooting DLLs, it also helps detect leaks and issu es, providing valuable insights into how the system works.

Process Explorer troubleshoots and handles DLLs effectively. To learn more about Process Explorer v17.05, read more
here
.

Caching
Although a cache is not a monitoring tool, it's important not to overlook them as computing relies heavily on caches, which e nhance data access speed and overall system performance. They store frequently accessed data for quick retrieval, making them essent ial for CPUs, SSDs, HDDs, web browsers, and web servers. Caches are smaller and faster than memory, acting as intermediate storage to optimize efficiency.

Linux autogrouping

In Linux, autogrouping optimizes desktop performance during CPU-intensive workloads by grouping processes and ensuring fair CPU cycle distribution. Autogrouping tells the process scheduler component in Linux to act based on a group's configured "nice level" instead of individual processes. However, autogrouping can interfere with traditional processes. When enabled, the "nice" value primarily affects priority within the group, reducing the effectiveness of "nice" and "renice" commands. Even programs setting their own nice levels may still receive a "fair" share of CPU time.

Pprofile - profilergnom
Cachegrind - get a cache, profiler

Kcachegrind - gui for profilers

More about improving our code
Software development is a key component of Information Technology, as it involves creating and maintaining applications that allow computer users to solve problems and accomplish tasks. As the digital landscape continues to evolve, software development plays an increasingly important role in creating new applications, enhancing existing ones, and maintaining the infrastructure that supports them. For this reason, it's important that you, as an IT professional, develop an understanding of software development. This understanding will enable you to identify and address issues quickly, devise effective solutions, and establish yourself as a trusted partner in your organization. In this reading, you learn how profiling tools help you to diagnose problems with the performance of your organization's applications.

Profiling
Software profiling is a diagnostic technique used to analyze real-time resource utilization of applications and monitor applications. This process involves examining key performance metrics like CPU utilization, memory consumption, and disk space usage. By dissecting these aspects, developers can gain valuable insights that guide performance improvements and optimization strategies. Additionally, benchmarking is a crucial practice in software development that involves a deep analysis of where your application spends its time and cycles. This is done by assessing the time and resources it consumes. It allows you to gauge code speed against a baseline and even against competing software or products. In Python, you benchmark through the Timeit module. This measures the execution time of your code segments, helping you pinpoint potential bottlenecks by conducting mini benchmarks for individual functions. In this way, you can improve your application's efficiency and optimize the code.

Profiling tools, such as Flat, Call-graph, and Input-sensitive, are integral to debugging. Developers can use these tools to generate detailed source code reports to understand how an application behaves and uses resources.

The importance of profiling extends beyond the development of software. It's also useful in the development of computer architecture and compilers. Through profile-guided optimization, developers can predict program behavior on new hardware configurations, enabling optimization algorithms to be refined and performance to be improved.

Over the past four decades, software profiling has evolved substantially. It remains an indispensable asset for programmers, computer architects, and compiler designers. Optimizing responsiveness and resource allocation helps developers craft high-performance software aligned with modern standards and expectations.

Profiles play an essential role not only in software development but also within the broader IT landscape. Profiling helps software engineers design efficient and effective applications by monitoring and analyzing real-time resource use. The ability to profile serves as an invaluable tool for IT professionals. While profiling isn't new, techniques such as this remain relevant today, and profiling provides a solid foundation for software development by improving responsiveness and optimizing resource usage.

Executor - used to control

Concurrency and parallelism
In Python, you can use concurrency to allow multiple tasks to make progress at the same time, even if they don't actually run simultaneously. This is useful when you want to optimize how tasks are scheduled and resources are used, especially for I/O-bound tasks. Concurrency lets you efficiently manage these tasks, ensuring they can smoothly move forward without being held back by other tasks.

Parallelism, on the other hand, involves running multiple processors or CPU cores at the same time. This is great for tasks that are CPU intensive. By dividing the work among multiple cores, parallelism can speed up these tasks significantly and reduce processing time.

By combining concurrency and parallelism in your Python programs, you can double their power. This should make your programs run more efficiently and responsively.

Concurrency for I/O-bound tasks
Python has two main approaches to implementing concurrency: threading and asyncio.

Threading is an efficient method for overlapping waiting times. This makes it well-suited for tasks involving many I/O operations, such as file I/O or network operations that spend significant time waiting. There are however some limitations with threading in Python due to the Global Interpreter Lock (GIL), which can limit the utilization of multiple cores.

Alternatively, asyncio is another powerful Python approach for concurrency that uses the event loop to manage task switching. Asyncio provides a higher degree of control, scalability, and power than threading for I/O-bound tasks. Any application that involves reading and writing data can benefit from it, since it speeds up I/O-based programs. Additionally, asyncio operates cooperatively and bypasses GIL limitations, enabling better performance for I/O-bound tasks.

Python supports concurrent execution through both threading and asyncio; however, asyncio is particularly beneficial for I/O-bound tasks, making it significantly faster for applications that read and write a lot of data.

Parallelism for CPU-bound tasks
Parallelism is a powerful technique for programs that heavily rely on the CPU to process large volumes of data constantly. It's especially useful for CPU-bound tasks like calculations, simulations, and data processing.

Instead of interleaving and executing tasks concurrently, parallelism enables multiple tasks to run simultaneously on multiple CPU cores. This is crucial for applications that require significant CPU resources to handle intense computations in real-time.

Multiprocessing libraries in Python facilitate parallel execution by distributing tasks across multiple CPU cores. It ensures performance by giving each process its own Python interpreter and memory space. It allows CPU-bound Python programs to process data more efficiently by giving each process its own Python interpreter and memory space; this eliminates conflicts and slowdowns caused by sharing resources. Having said that, you should also remember that when running multiple tasks simultaneously, you need to manage resources carefully.

Combining concurrency and parallelism
Combining concurrency and parallelism can improve performance. In certain complex applications with both I/O-bound and CPU-bound tasks, you can use asyncio for concurrency and multiprocessing for parallelism.

With asyncio, you make I/O-bound tasks more efficient as the program can do other things while waiting for file operations.

On the other hand, multiprocessing allows you to distribute CPU-bound computations, like heavy calculations, across multiple processors for faster execution.

By combining these techniques, you can create a well-optimized and responsive program. Your I/O-bound tasks benefit from concurrency, while CPU-bound tasks leverage parallelism.

Selecting the right approach
Before developing your program, it is essential to determine whether you want to incorporate concurrency, as it is generally easier to add it later than to remove it. In order to make this decision, you must understand the tasks your application needs to perform. Your approach will depend on whether your program is CPU-bound (processing) or I/O-bound (communicating).

When you need to wait for external resources, concurrency with asyncio or threading would be more appropriate. Taking advantage of idle time during I/O operations allows your program to handle multiple tasks concurrently.

On the other hand, if you're dealing with CPU-intensive tasks, such as compression, rendering high-definition videos, or running complex simulations, multiprocessing is a good choice. By doing so, you can ensure that your system performs well by taking advantage of the power of multiple processors. You can reduce processing time by distributing computational tasks across multiple cores.

To learn more about the differences between concurrency and parallelism and how to choose the appropriate approach based on your tasks, read more
here
.

Asyncio events and task loops
Python's asyncio library enables concurrent execution of multiple tasks through asynchronous operations using event loops and coroutines. A coroutine can pause execution while waiting for a specific operation, such as reading or saving data. Event loo ps are essential for scheduling and managing tasks, allowing smooth execution and reducing completion times. Unlike threading, this lightweight approach keeps long-running tasks from blocking the main application.

With Asyncio, you can efficiently handle small tasks, like sending emails or notifications, without creating many threads, re sulting in faster notification responses. When combined with aiohttp, asyncio effectively manages multiple API calls concurrently. Async io offers an efficient way to handle data input/output tasks, allowing developers to create high-performance applications through simultaneous task execution.

This tool optimizes Python programs, especially those that handle notifications, web requests, and data-related tasks. If you want to learn more about asyncio and its concurrent programming capabilities, read more
here
.

Terms and definitions from Course 4, Module 2
Activity Monitor: Mac OS tool that shows what's using the most CPU, memory, energy, disk, or network

Cache: This stores data in a form that's faster to access than its original form

Executor: This is the process that's in charge of distributing the work among the different workers

Expensive actions: Actions that can take a long time to complete

Futures: A module provides a couple of different executors, one for using threads and the other one for using processes

Lists: Sequences of elements

Memory leak: This happens when a chunk of memory that's no longer needed is not released

Profiler: A tool that measures the resources the code is using to see how the memory is allocated and how the time is spent

Real time: The amount of actual time that it took to execute the command

Resource Monitor (or Performance Monitor): Windows OS tool that shows what's using the most CPU, memory, energy, disk, or network

Sys time: The time spent doing system level operations

Threads: Run parallel tasks inside a process

User time: The time spent doing operations in the user space

Netstat -n -l -p
-l active listening
- p -process id

Computing is inherently intricate. For this reason, IT professionals must develop a comprehensive understanding of vulnerabil ities that include, but aren't limited to, hardware malfunctions, operating system glitches, and software deficiencies. As an IT profess ional, you'll become acquainted with viruses, malware, low memory, and constrained disk space, which can cause software and OS corruption. Professor Clay Shields at Georgetown University states that crashes are mostly caused by operating system errors. The spectru m of hardware failures, including disk errors, can cause irreparable harm, even with minor component degradation. Simultaneously, OS software errors include memory access blunders, perpetual loops, and buffer overflows. Operating systems can also plagued by problems like unstable drivers, memory leaks, and driver conflicts.

To learn more about the causes behind computer system failures, read more
here
and
here
.

Blue screen of death (BSoD)
A common disruption in computing systems is the kernel panic in Mac OS, also known as the notorious "Blue Screen of Death" (BSoD) in Windows, both of which require restarting the computer. Although rare, analyzing these occurrences is essential for uncovering OS issues. BSoDs are usually caused by hardware glitches, problematic drivers, or abrupt process terminations. These failure screens display error codes, memory locations, and technical insights related to the crash.

Reading system logs
System logs are crucial for understanding and resolving issues across multiple operating systems. Whether you're using a Mac, Linux, or another system, delving into these logs can yield valuable insights. Analyzing logs is critical for identifying system errors and crashes on Windows computers. The Windows logs such as System and Application carefully record data retrieval events, providing insight into software, hardware, and user interaction.

To learn more about reading the logs available in Windows, read more
here
.

Mac system logs provide insights into system operations. By using the Console app, you can capture error messages, warnings, and interactions between hardware and software. These logs are especially useful when investigating system behavior.

To learn more about system logs on a mac, read more
here
.

Linux system logs offer insights into troubleshooting. They give detailed information about the Linux environment, such as errors and hardware-software interactions. Using command-line utilities, you can access these logs to identify unusual behavior patterns. These logs provide a holistic overview of system performance.

To learn more about the system log on Linux, read more
here
.

Process Monitor
Monitoring tools like Process Monitor in Windows provide real-time visibility into file system actions, registry changes, and process behavior. With features from legacy tools such as Regmon and Filemon, Process Monitor captures input/output parameters, uses non-destructive filtering, identifies root causes, and compiles comprehensive process data. This includes details such as image paths, commands, user information, and session IDs. The tool offers customizable columns, flexible filters, and scalable logging to enhance event management. Tooltips provide quick access to log files and reveal process relationships. It also records boot-time operations for comprehensive tracking, troubleshooting, malware detection, and system activity analysis.

To learn more about the Process Monitor, read more
here
.

Linux strace command
You can use a Linux strace command to trace system calls and signals. It aids in debugging and diagnostics by analyzing application and process behavior. It captures system calls, pinpoints issues, optimizes code, and enhances system performance. You use strace by entering the program's name and any arguments at the command line. This tool logs detailed system call information, enabling you to analyze bottlenecks, unintended behaviors, and misconfigurations. The strace command contributes to a better understanding of OS and application interactions, ultimately leading to efficient software development and effective issue resolution.

To learn more about the strace command, read more
here
.

Tracing system calls
Tracing system calls on Linux is useful for identifying security risks and tracing system calls, which reveal the intricate interactions between processes and operating systems. You can trace a Linux system call using the ptrace API and the strace command, and you can trace a Mac OS X system call using the dtrace system. Windows uses the GUI tool Process Monitor, and additional projects enhance system call tracing. Tools like Logger, LogView, and NtTrace leverage Microsoft's Event Tracing for Windows (ETW) capabilities. Across operating systems, tracing system calls remains pivotal for development and monitoring, anchoring system analysis and optimization.

To learn more about tracing system calls, read more
here
.
Valgrind - seg fault checker
Dr. memory for windows

Ulimit - get core fiesl.

Gdb - view core files

Pdb3 - python debugger

\Assertions are logical tests that developers use as a sanity check when writing code. In Python, you use an assert statement to write these sanity checks. When you write an assert statement, it is important to write it with one thing in mind: The condition yo u include with the assert statement should always be true. If the condition is false, you can use this information as a main indicator that the program has a bug. If the assert statement is false, it will automatically terminate the execution of the program and will di splay an error message. At this point, you can correct or fix the bug before continuing to write code to ensure you don't introduce any addi tional bugs.

Pdb commands:

n (next) is just one of the commands you can use to navigate the debugger. Other commands include:

a (args): Show the arguments of the current function.

b: Manually set a persistent breakpoint while in debugger.

n (next): Execute the next line within the current function.

s (step): Execute the current line and stop at the first possible occasion (e.g., in a function that is called).

c (continue): Resume normal execution until the next breakpoint.

p (print): Evaluate and print the expression, e.g., p variable_name will print the value of variable_name.

Pp (pretty-print): Pretty-print the value of the expression.

q (quit): Exit the debugger and terminate the program.

r (return): Continue execution until the current function returns.

tbreak: Manually set a temporary breakpoint that goes away once hit the first time.

!: Prefix to execute an arbitrary Python command in the current environment, e.g., !variable_name = "new_value" will set variable_name to "new_value".

Terms and definitions from Course 4, Module 3
Breakpoints: Debugging features that lets code run until a certain line of code is executed

Communications lead: The lead person who needs to receive timely important communication updates

Core files: Files that store all the information related to the crash to debug the issue

Incident commander (incident controller): The person who needs to look at the big picture and decide what's the best use of t he available resources

Pointers: The variables that store memory addresses

Postmortems: Documents that describe details of incidents to learn from mistakes

Undefined behavior: The code is doing something that's not valid in that programming language

Valgrind: A powerful tool that can tell if the code is doing any invalid operations, no matter if it crashes or not

Watchdog: This is another process that checks whether a program is running and, when it's not, starts the program again

Watchpoints: Debugging feature that lets code run until a variable or expression changes

Wrapper: A function or program that provides a compatibility layer between two functions or programs, so that they can work well together

Od - dump files in octal and other formats

Decorator: Used in Python to add extra behavior to functions without having to modify the code.

You've learned a lot about how to avoid memory leaks and manage your OS and network to achieve the best performance. Let's look at a few more ways you can speed up your application processing and troubleshoot memory and system performance.

Using concurrency and threading
Python has several tools to streamline the processing of code, including threading, multiprocessing, and a library called
asyncio
.

With these, you can run things (threads, tasks, and processes) in an overlapping fashion, called concurrency. Similarly, you can use asynchronous threading, where you can tell the OS to prioritize certain threads over another. Although these are similar, they each have their own advantages.

You can also learn about concurrency and threading in the reading:
More about complex slow systems
.

Concurrency
Depending on your application's resource needs, you may be able to use concurrency to help speed up things that are slowing down processing. Concurrency involves adding a bit of extra code to your programming to allow it to run multiple things in a sequence, but with overlapping time frames. For example, you could have hundreds of threads or requests, but you can set them to run on top of each other, instead of waiting for one to complete before the next starts.

In I/O-bound programming, where there is a lot of interfacing with networks or other hardware, using concurrency can help download network-based content faster and more efficiently. In CPU-bound programming, where the program is busy processing data, you can use concurrency to spread heavy CPU processes across multiple processors, essentially splitting the heavy load among workers.

Keep in mind, however, that adding concurrency to your code means … that's right: more coding, and that introduces a greater risk of introducing hard-to-find errors into your program.

Learn all about concurrency, different forms of multitasking, ways to enable concurrency outside of asyncio, and much more in this in-depth and helpful
article
. And, for a good overview with clear examples of what concurrency is, read this
article
.
Checking memory usage
Memory leaks can be a big problem when coding because they slow down the processing of your application and may create crashes. When you can evaluate programming to find issues, you can then streamline your code and/or processes to release code cleanly.

Although Python automatically manages memory as part of its language, there are several other tools you can use to look for memory leaks to ensure yours is running as efficiently as possible.

If you're looking at a section of your code that you suspect may be slowing your application, there are packages, such as
memory-profiler
, that will evaluate a single function or code, line by line, to show detailed memory use. You can also use it to evaluate your application's memory usage over time, helping identify memory that isn't releasing as regularly as it should.

If you need to look at the application as a whole, try
guppy
, a library to view and evaluate memory use by different object types.

For more about memory-profiler and other tools, read
this article
.

Checking for network problems
If you’re not a network administrator or engineer, all those acronyms and their configurations can be a bit of a mystery: IP, SASE, IMAP, MAC, SSH, DHCP … If you’re a programmer facing network-connection issues, it can be overwhelming to sort out.

Lucky for you, you have some in-built tools to help you identify the source of the problem.

The first thing to try is to ping the server to see if it’s a server issue or an actual network issue. Sending a simple telnet command to the server and port you’re trying to reach can tell you if the server is having a timeout issue (or is completely down), or if there might be a problem accessing it for some other reason.

If you can’t get a response from the server, then you can probably assume there is something wrong with it or the network in between. Here’s where you detective skills come into play. First, you’ll need to test to see if it’s something on your end or on the receiving end.

Check your default gateway using the ipconfig /all command (in Windows) or ifconfig -a (in Linux). This will show you the IP addresses and DHCP connections. If you can’t see a DHCP, then you either need to renew its lease (the network connection between you and the IP address) or, in some cases, the DHCP server is down, which is a larger problem for the networking experts.

If you can see the DHCP, but the connection still isn’t working, you can try testing various devices to pinpoint the issue. In Linux, you can use the #arp -n command to see a list of MAC addresses for devices on the network. MAC addresses are like identifiers for individual computers instead of a network (the IP address). When you use #arp -n, then, you may be able to see if a device is missing a MAC address, which would tell you that it is down.

For more examples and details, check out this
article
. You’ll want to keep it bookmarked!

Key takeaways
Many things may cause performance issues in your programming, from memory leaks to network issues, to slow processing times. By knowing some simple tricks to code, scan, and troubleshoot, you can help your program run efficiently as possible.

Terms and definitions from Course 4, Module 4
Bandwidth: How much data can be sent or received in a second

Centralized logs collection: This means there's a special server that gathers all the logs from all the servers, or even all computers in the network

Decorator: Used in Python to add extra behavior to functions without having to modify the code

Exhausted: When resources are used completely and programs are getting blocked by not having more access to those resources

Garbage collector: A tool in charge of freeing the memory that's no longer in use

Latency: The delay between sending a byte of data from one point and receiving it on the other

Memory profiler: A tool used to figure out how the memory is being used

Reproduction case: A clear description of how and when the problem appears, a way to verify if the problem is present or not

Swap: A space in the hard drive where the operating system puts the parts of the memory that aren't currently in use

Technical debt: The pending work that accumulates when a quick-and-easy solution is applied instead of a sustainable long-term one

Traffic shaping: This is a way of marking the data packets sent over the network with different priorities, to avoid having huge chunks of data use all of the bandwidth

Interviewing:
t can be difficult to land interviews off of a resume alone. There are strategies that can make a big difference in getting r esults. This advice can help you overcome some of the challenges associated with job boards, such as out-of-date listings and heavy application volumes that can make it hard to stand out.

The idea of networking can seem daunting for many, however there are actionable steps you can take to make the most of your networking efforts. You'll learn how to find people to connect with, how to schedule and prepare for important conversations, what to talk about, and how to follow up. If some of these strategies and actions feel challenging at first, don't worry; they get ea sier over time. Plus, you'll be getting more interviews, so it will all be worth it!

The Importance of Networking
Learning how to network effectively is a really valuable skill with a wide range of benefits. It's something you'll want to c ontinue to focus on, and the connections you make and maintain through strategic networking can have long-lasting positive effects on your career advancement.

Strategic networking can help you overcome some of the challenges associated with online job applications. Some of the benefi ts of networking include:

Getting accurate information about job availability. The fact that a role is posted online doesn't mean that the hiring team is actively reviewing applications. There is often a delay between the time a role is open and the time it's posted online, as well as th e time it is filled and taken down from online job boards. At the same time, there are often open positions that are not (yet) posted onli ne for a variety of reasons. Networking can help you ensure you've got up-to-date information.

Learning more details about the role. Job descriptions are not always precise. As a result, you might end up applying for rol es that you think are a good fit but are, in fact, not. Or, you might fail to properly tailor your application to meet the needs of the h iring team. Insider information via networking can help you understand what the team is really looking for.

Standing out amid the competition. Once a job is posted to a job board, there are often tens or even hundreds of people apply ing to it. It can be difficult to stand out. Networking can help you get an early jump on a new opportunity before it's posted.

The networking process described in this guide can help you address all these challenges.

Through a short and focused conversation with someone at your target company, who has insider knowledge of relevant opportuni ties, you will be able to:

Understand the requirements for your target role at that specific company. Jobs with the same title can vary greatly from com pany to company, and the actual requirements are not always obvious from job descriptions.

Gain insight into the company's organizational structure and team culture to learn what's required for success, and understan d how best to position yourself in your application materials and interviews.

Learn about ways to monitor and apply for opportunities at that specific company, so you can know exactly what's available an d how to float your application to the top of the pile. You might even be able to get a referral.

Establish a relationship with a professional who might be able to help you in your current job search and be a part of your p rofessional network moving forward.

Networking for Your Job Search: The Process
How to connect with the right people

The process of networking for your job search begins with identifying the right people to network with. You are looking for i nsider information on the role and its application process, as well as other relevant opportunities.

People you select to network with should work in, or close to, your target role at a company you are interested in working fo r. These people will have the information you need, beyond what's publicly posted online. They will likely understand the exact skills and qualities the hiring team is looking for. They may know the status of currently open roles and upcoming openings, and they mi ght even be able to connect you directly to the hiring team.

If you are already connected to the right people, you can jump straight to Step 2 below. If you don't currently know such peo ple, begin with finding and connecting with them as described in Step 1.

Step 1: Finding the right people

To begin, put together a list of the companies you're interested in. The more companies you have on your list, the more people you will be able to reach out to, and the more opportunities that will be available to you. Don't be surprised if your company list grows to 50 or more companies. It might sound like a lot, but remember that not every company will have the right role available when you need it.

If you are not sure how to identify target companies for your job search, consider the following ideas:

Search job boards for openings. If a company has ever posted a relevant role, it's worth exploring further.

Go through your existing contacts and research the companies where they work. Even if you don't know anyone in your target role, your personal and professional contacts might be able to introduce you to their relevant coworkers.

Identify a target industry (e.g., education, technical, finance). If you know one company within that industry, you can perform a search for its competitors to expand your list.

Map out companies located in your area (or companies with a remote workforce, if you are looking to work remotely). You want to make sure you can definitely be considered for any opportunity you uncover.

Remember, your target companies do not need to have open jobs posted—you will find out exactly what is currently available there through your networking conversations.

Step 2: Connecting via LinkedIn

Once you know the companies you are interested in, you can start connecting with relevant people. The method described here uses LinkedIn because it is accessible to most people. You can also ask for introductions from mutual connections, attend professional events to meet people, post in networking communities online, or use any other way you prefer.

To find people through LinkedIn, begin with performing a LinkedIn People Search using your target job title as the search string, and setting a filter for "Current Companies." See below for an example searching for a Data Analyst at Coursera.

Review the profiles that come up to identify people you want to reach out to. Focus on people you'd like to learn from and that you think you can build a rapport with based on their background, interests, and even their tone of communication. Keep in mind that people with well-developed LinkedIn profiles—that include profile photos, summaries, and other details—are more likely to reply to you than those who have very basic profiles, because they are likely to be more active LinkedIn users.

Once you identify a person you might be interested in speaking to, send them a connection request with a note explaining why you are reaching out.

Sample LinkedIn outreach message:

Hi <name>, I discovered your profile because of the interesting work you do as a <role> at <company>. I'd appreciate an opportunity to ask you a few questions to learn more about what you do and what it is like to work at <company>. Thank you in advance for connecting with me!

Note that some of your connection requests may go unanswered. Don't get discouraged or take it personally. Many people are too busy or simply don't monitor their LinkedIn messages. The great thing is that LinkedIn provides you with access to a large number of professionals, and it's a great idea to reach out to a lot of people.

Step 3: Schedule and prepare for the conversation

Once you've established the connection, you can ask your new contact for a time to speak. It is important to be open to communicating via the connection's preferred approach (in-person, video, phone, in writing, etc.), but ideally, you want to schedule a live conversation. It's generally a more effective way to build a relationship, and can often make it easier to get your specific questions answered.

Make scheduling easy by suggesting a specific time to speak, offering to work around their calendar, and sending out a calendar invitation with information on how you will connect (phone, video conferencing, etc.).

Sample meeting request message:

Thank you for accepting my connection request! As I mentioned, I reached out because I'm researching <industry/company> and would really appreciate an opportunity to ask you a few questions about your experience in <role, company>. Would you be open to scheduling a 15-minute video or phone call on <date, time>? I'm also happy to adjust to your schedule if you prefer another time.

Note that some people find it easier to provide information in writing. If you don't get a response to your original request for a conversation, you can follow up by asking whether it would be easier for them to answer a few questions over email. Remember, everyone is different and it's important to gauge and adjust to the style of the person you are reaching out to!

Don't be discouraged if someone does not reply to you immediately. People are busy. Since you have already established a conn ection, it's a good idea to follow up after a few days, and then again a week later to give them a chance to reply.

If you still don't hear back after a couple of follow-ups, you can assume this person is too busy at this time to speak with you and move on to other potential contacts. Remember that while this is a process of developing personal connections, it's also a numbers game, and you should plan to reach out to a lot of people!.

Before moving on, acknowledge your decision to your new contact—a quick note will help ensure there is no awkwardness so you can easily reconnect in the future.

Sample moving-on note:

I'm sorry we haven't been able to connect. I definitely don't want to flood your inbox with requests, so I just wanted to tha nk you again for connecting with me, and if you do end up having some time to chat, please let me know.

Preparation

Once the conversation is on the calendar, it's time to prepare. Remember, your focus should be on learning about your target role at the company and determining the best ways to connect to new opportunities. Things you'll want to focus on include:

What is the day-to-day like in the role? What is the team structure, how are priorities decided, what do they like about their work, and what do they struggle with?

What skills and experiences do the hiring team look for? What is essential, and what is nice-to-have?

Do they think your skills and background are a good fit for the role, or are there ways you can improve your candidacy throug h education or experience?

What is the best way to monitor and apply for opportunities? Is there anything coming up that is not yet posted on the career s page?

Are there any other people they can recommend that you speak with?

To inform your questions, you'll want to conduct thorough research on the person you are speaking with, the company they work at, and your target role. Consider the following sources of information:

Your contact's LinkedIn profile, and any information it links to. Look for information to inform your questions as well as an ything that can help you build rapport, such as shared volunteering interests, hobbies, school experience, etc.

Job descriptions for your target role at the company (if available). During the conversation, you'll have an opportunity to c larify requirements and responsibilities.

LinkedIn profiles of people working in your target role at the company. You want to understand their skill sets and backgroun ds to get additional insights into what it takes to succeed in this role.

Company website. You should have a good understanding of the company's mission, business, and anything else they chose to hig hlight to the public.

Company reviews on platforms such as Glassdoor. It's a great idea to see what people are saying about the company, so you can ask more specific questions about the culture.

News about the company. Just in case there is something significant happening at the company, you want to be aware of it.

Company careers page. Make sure you know which roles are currently posted so that you can ask about the status, and about app lying to them directly.

Step 4: Speak with your new contact

Speaking with strangers does not come naturally to many people. If you are feeling uncomfortable before or during your first few conversations, that's completely normal! It will get easier with time as you develop the invaluable skill of networking.

Remember that the other person is also going into a conversation with a stranger (you) and might not know what to expect. To make both of you comfortable and to help build rapport, be ready to set the structure for the conversation.

Remind them about who you are, why you reached out, and what your goals are for the conversation. By this point, you will have done extensive research in preparation for the conversation, but your new contact might not have had the time to look at your profile and doesn't know why exactly you reached out. Help them out by starting with a brief overview of your background and the reasons for the conversation.

Monitor time. Conversations like this generally last 15–30 minutes. Make sure you respect the other person's time by keeping the meeting to the length you had originally agreed upon, unless the other person wants to continue talking.

Make it about them. While you are there to learn, the person you are speaking to is being generous with their time, and it's your responsibility to make them feel valued and appreciated. Explain why you wanted to talk to them and show the research you've done. Honest praise and genuine engagement go a long way.

Listen more than talk. Since you are there to learn about their experience and company, the primary focus of the conversation should be on the other person. Some people might be more talkative, while others may need more input from you in order to engage. Ideally, they should be speaking for 50% to 80% of the conversation. Don't be afraid of short pauses, and be respectful and patient if they need time to gather their thoughts.

Take note of action items as you go along. There are many action items that can come out of a conversation like this: you might need to send the other person your resume, they might offer to connect you with someone else, either one of you might want to share articles or resources that come up in the conversation, etc. It's your responsibility to keep a record of these action items, so you can follow up on your promises and make it easy for the other person to remember theirs.

Close the conversation by clarifying what's next. Thank them for their time, summarize what you have learned, and go over any action items from the conversation. The goal is to make the other person feel useful and appreciated—after all, they've been generous with their time.

Asking for a referral

Getting a referral is an ideal outcome for a networking conversation. However, not every conversation will end in a referral. Sometimes, there will be no role available or the person might not be open to referring you for a variety of reasons. Make sure not to take this personally or push too hard. Their reasons may have nothing to do with you specifically. It's important to respect their boundaries and comfort levels. It is also important to go into the conversation without the expectation of a referral. Focusing on learning about the role and getting advice from your new connection will take the pressure off you and them.

If, during the course of the conversation, you confirm that there is a role available that you are qualified for, do consider asking for a referral. You should be able to sense from the conversation whether the person thinks you could be a valuable addition to their team and therefore open to referring to you. If you have any doubts about that, provide an easy way for them to say "no" to you to avoid an awkward situation. For example, you can ask, "Would you be able to refer me to this role, or do you recommend I apply online?"

If your contact agrees to refer you, make sure you understand exactly what's required from you. Depending on the company's system, you might need to apply through a special referral link, have your contact submit your resume internally on your behalf, or apply online and then have your contact reach out to the relevant member of the hiring team.

Step 5: Follow up

Always send a thank-you email within a day or two to the person who has been generous enough to share their time and expertise with you. Go beyond the basic "thank you" and reinforce the connection you've made by:

Reiterating what you have learned

Following up on your action items from the conversation. Include any materials you had promised to share and list out what else you are going to do based on the conversation (make sure to follow up on those as well when the time comes!)

Gently reminding then about any action items the other person had volunteered for

Offering to repay the favor by sharing any information that might be valuable to the person, or offering to connect them with people in your network

Sample thank-you note:

Hi <name>,

It was great to catch up with you today and hear about the incredible work you are doing at <company>, and I was excited to learn about our shared interest in <x>. Here is a link to the article I had mentioned on <topic> that I thought you might enjoy.

Thanks again for sharing about the <role> opening with me and sharing my resume with the hiring manager! My resume is attached. Please let me know if you have any questions or need anything else from me.

Again, it was great to speak with you. Thank you for your time and willingness to share your experience with me! Please let me know if I can ever be of any help. I have a pretty extensive network in <industry> and would be happy to introduce you to any of my connections.

Some conversations naturally lead to ongoing relationships where people find a lot in common and naturally stay in touch, while others don't create enough rapport to solidify the connection. Even if your conversation falls into the second category, as long as you feel that you'd like to keep this person in your active network, there are actions you can take to develop the connection over time. The key to developing your new connection is finding natural touchpoints moving forward. For example:

Share updates on your job search. Follow up on any advice from the conversation once you have a chance to act on it. Your connection will appreciate that you valued their guidance and will be glad to know if it helped. Also, remember to update and thank them once your job search is complete.

Send interesting information as it comes up. If you come across an article or information that reminds you of the person, it is a great reason to send them a quick note.

Engage on LinkedIn. If your new connection is active on LinkedIn, commenting on their posts and updates is a great way to continue the conversation.

Add them to your celebrations calendar. Add them to your holiday mailing list. In addition, if any important dates, such as a birthday, come up in the conversation, make sure to mark your calendar and send your congratulations.

Do be mindful about your rate and volume of outreach, as you don't want to overdo it. Make sure to establish a pace that feels right for the relationship.

Continue Growing Your Network
You now know how to find, reach out to, and develop relationships with people who can help your job search through insider information. Not every conversation you have will result in an immediate job lead, but many will. Networking is the most reliable way to get interviews, and it's available to everyone with a LinkedIn account, effective strategies and some grit.

Don't be discouraged if you don't feel great about your first few conversations, or if they don't result in referrals. It is normal to feel uneasy about speaking with strangers, particularly at first. It's a skill you need to practice. Each conversation you have with an industry professional is a win. You are building one of your most valuable professional assets—your network—one person at a time!

Companies
You have been exploring the interview process in IT Automation. No two companies are alike, so you will need to research each company with which you interview. As part of your research, you should learn the size of the company and how long it has been in business. This is because the interview process at larger, more-established companies can differ greatly from that of smaller companies and start-ups. In this reading, we will discuss the common differences between interviews at these types of companies. Knowing these differences can help you prepare and know what to expect, which can ultimately help you make a great impression.

Who conducts the interview
Established companies often have well-developed human resources (HR) departments. Interviews at these companies are likely to be designed or conducted by experienced HR professionals. The application process might involve a series of interviews—from a preliminary interview with an HR associate to interviews with potential teammates to an interview with the actual manager you would work for if you were hired.

Startups and smaller companies, however, may not have an HR department at all. Applicants are likely to be interviewed by someone high up in the company, such as senior leadership or the CEO. Startups are also unlikely to require as many rounds of interviews as larger companies due to limited resources and time constraints.

Structure of the interview
Interviewers at established companies will usually ask questions that relate to your technical skills and competence level. These companies want to determine if you will fit well into the available position. The interviewers will generally ask you a series of common interview questions about your experience, training, and how your past roles have prepared you for a position in a company of their size.

At startups and smaller companies, on the other hand, the interviewer may ask a few questions and then allow you to steer the interview's direction. Interviewers at smaller companies may also pay more attention to your ability to fit into the company culture. Additionally, since these businesses often require that people perform their jobs with less supervision than at more established companies, they tend to seek employees who are independent and proactive. Being able to actively drive the conversation with your own ideas can help display these qualities.

Expectations for the role
The interviewer's expectations for the role might differ based on the company's size, and you will need to emphasize different competencies based on these expectations. At a small company, for example, you may be expected to take on many roles—perhaps acting as the entire marketing department. At a larger company, you might be asked to mostly perform one task that you do exceptionally well on many projects.

At a larger company, there may be more established processes to follow, and you may often need to get approval from stakeholders to move forward on tasks. At a smaller company, you might have fewer approvals processes to navigate, but you will need to be able to execute tasks with little guidance.

One way you can demonstrate these different competencies is in the way you respond to behavioral interview questions. These are types of questions that require you to share a time when you were faced with a particular situation or had to use a certain skill. In an upcoming activity, you will practice a strategy for responding to these types of questions.

Length of the process
Established organizations may require you to go through several interviews before making you a job offer. They do this because they can generally afford to take time to find the best fit for the role. These companies may take three weeks or more to screen and interview applicants.

Startups tend to have a much quicker interview-to-hire process—generally around two weeks. Because people higher up in the company conduct the interviews and make the hiring decisions, not as many rounds of interviews are typically required. Additionally, since smaller companies and startups may have fewer employees available to get the work done, they often aim to fill open positions quickly.

Level of formality
Interviews at established companies tend to be more formal and structured than interviews at startups. Since the interviewers generally want to assess all the applicants on the same basic criteria, the process can seem somewhat conventional. Your behavior, dress, and the language you use is generally expected to be more formal in interviews at these companies.

Interviews at startups tend to be more casual. The structure of the interview is looser, and the dress, behavior, and language you are expected to use may be less formal, as well.

Key takeaways
The norms discussed in this reading can vary from industry to industry and company to company, so be sure to research the specific company you are interviewing for ahead of time to learn what to expect. This will prepare you for the different aspects of the interview, such as what to wear and the kind of questions you may be asked. But regardless of the company's size or how long they have been in business, you should always be prepared to share your qualifications and skills, convey what you know about the company and the role, and describe why you would be an excellent fit for the position.

Developing an Elevator Pitch
When interviewing with potential employers, it's important to communicate who you are, your value as a Python Developer or Automation Engineer, and what you're searching for in a job. A simple way to do this is with an elevator pitch. An elevator pitch is a short, memorable description that explains an idea, business, or service in an easy-to-understand way, typically in 60 seconds or less (the average amount of time of an elevator ride).

While an elevator pitch is usually specific to an idea or a product, you can also use it to sell yourself as a professional to potential employers. In an interview, a strong elevator pitch can be used to stand out to your interviewer. It can be used to help explain why you're a good fit for the role, or to answer the popular interview question, "tell me about yourself." This reading helps you prepare your elevator pitch to sell yourself and the value you can provide as a Python Automation professional.

Provide an introduction
Start by providing an introduction. Introduce yourself and give a brief overview of your professional background. Explain som e job roles you've had, your years of work experience, and the types of industries you've worked in. If this is your first Python automat ion job, mention some of your past roles with transferable skills. Even if you're interviewing for your first internship or job in Pyt hon automation, it's important to clarify that this is what you want to do as a career. For example, you could say, "I want to ap ply my excellent technical problem-solving skills to find and implement automation solutions."

Show your excitement
This is where you share your passion for the field and why you want to work in the industry. If you're motivated to sell prod ucts online, mention that. This is also a good time to talk about your goals. For example, you could say, "I love creating automation solu tions because it gives me the opportunity to develop new Python functions. Long term, I'd love to develop my knowledge of automatio n for cloud management."

Communicate your interest in the company
Communicating why you are interested in the company—and not just the role—is a great way to help the interviewer recognize that you are knowledgeable about the company. For example, if you were interviewing for a position for Google's Software Developme nt team, you could say, "Google's Software Development team has the opportunity to work on software applications that the world uses. As a long-time Google App user, I'm looking forward to the opportunity to be a part of that mission and provide outstanding Python support."

Elevator Pitch Examples
To bring the structure of an elevator pitch to life, check out three examples of elevator pitches. The first is by Dan, a Sof tware Developer. The second is Sean, a Marketing Manager on the Google Ads team. The third is by Joi, an Associate Product Marketin g Manager.

Dan
Software Developer

I'm Dan, a software developer with 8 years experience developing solutions for logistics. The most recent solution I develope d is a database system that prevents large product shipments from being misdirected to the wrong destinations. Outside of my work wi th logistics, I develop AI software for websites and create websites for clients using that technology. My passion for software development and problem solving is what led me to some of the top logistics organizations. I love solving problems and helping my clients save time and money.

Sean
Marketing Manager, Google Ads

I'm Sean, a Marketing Manager for Google Ads, with over a decade's worth of experience in the field of digital marketing, mos t of that with Google.

When I went to school, I didn't even know this industry existed. I majored in English because I liked reading and writing. My first employer in digital marketing took a chance on me because of my experience with client management and spreadsheets, and they figured they could teach me about digital marketing. I'm glad the industry and I found each other.

Google is always innovating, which means when you work in this field you never stop learning. My first company actually ran a blog all about the latest changes to Google Ads (then called AdWords), and because of my English degree, I took a keen interest in the blog. In a few months, I was managing the blog, and it was through my posts on that site that Google found me.

It's been wonderful to be on the team that announces the latest changes and updates to Google Ads. Because of my hands -on experience buying ads myself, I can immediately see how somebody's workflow will change after an announcement. I love being a ble to tell Google's story to our advertisers so that companies of all sizes can continue to find success and grow their businesses.

Joi
Associate Product Marketing Manager

I'm Joi, an Associate Product Marketing Manager at Google with 10 years experience as a content creator for YouTube and organ ic social channels.

Outside of work, I run my own beauty e-commerce business, an experience that has helped me develop a plethora of skills around digital marketing and paid advertising, project management and operations.

My entrepreneurial mindset paired with my love for creativity is what led me to a company like Google. I thrive in ambiguity and love strategizing and solving problems from the ground up.

Key Takeaways
Creating a 60 second or less elevator elevator pitch is a great tool to use to quickly share who you are. Use an elevator pitch to introduce yourself to career and business connections in the future. You can even use your elevator pitch in other types of situations, like meeting new friends or new colleagues.

Completed

Asking the Interviewer Questions
In addition to an interviewer asking you questions, it's important that you ask the interviewer questions as well. Asking questions helps you learn more about the role and it shows your interest in the role.

In this reading, we list several questions you should consider asking your interviewer, and explain both the intention of the question and the benefits of asking it..

Why ask your interviewer questions?
One reason to ask your interviewer questions is that it helps you determine if you are interested in the role. One mistake people make in interviews is believing they are the only one being interviewed. Remember, you are also interviewing the organization to determine if you would like to work there! Ask questions to help determine if the organization is a good fit for you.

Another reason to ask questions is that it shows your interest in the role. When possible, make your question specific to the company you are interviewing for. For example, imagine during your pre-interview research, you come across an article discussing the company's entrepreneurial culture. You can mention that you read about the organization's entrepreneurial culture. Then, ask how that culture gets represented in the company.

When to ask your interviewer questions?
Often, at the end of the interview, the interviewer will ask you if you have questions. This is the perfect time to ask any questions that were not addressed in the body of the interview. Or, the interview may end without time reserved for questions, and that's OK, too. It's typically best to respect the interview time frame rather than ask questions past the time.

If the interviewer doesn't confirm they will allow time at the end for questions, one way to fit them in before time runs out is to ask during the interview. When asking during an interview, ensure the questions don't disrupt the flow. For example, if the interviewer mentions available training for the role, you can comment that you are interested in the company's training. You can then ask them what type of training is available for the position.

Additionally, if you are unable to ask any questions during the interview, you can follow-up with an email. Make sure your questions are directly related to the role and related to something you are genuinely interested in.

Example questions to ask your interviewer
How do you evaluate success in this role?

This question helps you better understand what skills or qualities make someone successful in the role. If the interviewer mentions skills or qualities you have, you can then discuss how you applied them in your previous experience.

Can you describe the typical day of someone in this role?

It's important to know the day-to-day activities of the position. Does this match with the type of role you're interested in? If it doesn't, the role may not be a fit for you. This question also confirms that the tasks for the role match the job description.

How would you describe the company's culture?

A company culture is the attitudes and behaviors of the company and its employees. Asking this question helps you better understand if the company's culture is a fit for you. For example, if you'd like to work for a company that supports creativity and encourages new ideas, look for that type of information when someone describes the culture.

What do you like about working here?

Similar to the question about culture, this question provides the positive qualities of a workplace. Ensure these qualities match with what you're interested in for a work environment.

Is there any training for the role and how is the training delivered?

If you're interested in receiving training for a role, consider asking this question. Additionally, you may want to ask how the training will be delivered, such as digitally, in-person, shadowing a current employee, or another method. Shadowing is when you closely observe another employee perform the role.

Do you have any questions or hesitations about my qualifications or experience?

If you ask this question at the end of an interview, it gives you a chance to address any concerns the interviewer may have about your work background. Sometimes the interviewer is interested in a particular experience that you may in fact have, but didn't include on your resume. This is the perfect question to address that discrepancy.

Key takeaways
When interviewing, you should ask questions to learn more about the organization and show your interest in the role. When doing pre-interview research, write down any questions you may have for the organization or the role. It's a best practice to have at least four questions prepared before the interview. If there is time available and the question seems appropriate, ask it!

Mod 5:

This has been a big module! We've covered a lot of ground related to cloud computing. We started by learning about clouds — not the ones that float through the sky, but computer services that run in a data center or remote servers that we access over the Internet. These clouds have different service types, from a bare bones environment that gives you the computing power to run your own software with your own configuration settings, to those that deliver a whole application or program to a user, such as easy to use email solutions like Gmail, storage solutions like Google Cloud, or productivity suites like Google Workspace.

Next we learned how to deploy a virtual machine (VM), made sure the VM was set up to serve our web app, and that it would stay updated via Puppet. A single VM can be useful for small operations with lower technical requirements, but as technical requirements increase, an organization will need to deploy more and larger cloud solutions. Luckily, it's easy to scale in the cloud. So we turned that single VM into a customized VM template that could be used to clone that VM as many times as we want, so scaling our cloud deployments would be easy.

When we had a VM template, we looked at different ways to interact with the platform. We saw how both the web interface and the command line tool can be used to create VMs in the cloud, modify their configuration, and control other things, using tools which are very effective at a small or medium scale.

At a large scale, you'll need to automate cloud deployments even further using orchestration. We looked at how tools like Terraform allow us to define our cloud infrastructure as code, which can give us a lot of control over things like how the infrastructure is managed, and how changes are applied. Orchestration lets us combine the power of infrastructure as code with the flexibility of cloud resources.

Then it was time to look at some of the options for building software for the cloud. FIrst, we looked into the different types of storage available, and what to consider when deciding which storage solution to use. These considerations include how we want to request data from storage, and how fast we want to be able to access the requested data.

Maybe the number one reason for using cloud computing is how much computing power is available across many servers. But in order to distribute our service evenly across however many instances we have available, we use load balancing. We talked about the different methods load balancers use to distribute the workload, and how they can monitor server health to avoid sending requests to unhealthy servers.

There is no doubt that computing methods are constantly changing, and this happens pretty fast.

We talked about how change management allows us to make changes in a safe and controlled way. We looked at how continuous integration (CI) can build and test code every time there is a change, and how continuous deployment (CD) can automatically control the deployment of new code to a specified set of rules. And we talked about different environments where new code could be tested before being pushed to production.

And we talked about limitations. Limitations you may come across when running services in the cloud are not the same as limitations when running services on physical machines, and you should take time to understand the limitations of any cloud solution you may choose.

Basically, you just learned a lot about cloud computing in a short period of time. And there's more to come. First, you've got a graded assessment that covers the material from this past module. Then it's time to start learning about deploying applications to the cloud.

Terms and definitions from Course 5, Module 1
A/B testing: A way to compare two versions of something to find out which version performs better

Automatic scaling: This service uses metrics to automatically increase or decrease the capacity of the system

Autoscaling: Allows the service to increase or reduce capacity as needed, while the service owner only pays for the cost of the machines that are in use at any given time

Capacity: How much the service can deliver

Cold data: Accessed infrequently and stored in cold storage

Containers: Applications that are packaged together with their configuration and dependencies

Content Delivery Networks (CDN): A network of physical hosts that are geographically located as close to the end users as possible

Disk image: A snapshot of a virtual machine's disk at a given point in time

Ephemeral storage: Storage used for instances that are temporary and only need to keep local data while they're running

Hot data: Accessed frequently and stored in hot storage

Hybrid cloud: A mixture of both public and private clouds

Input/Output Operations Per Second (IOPS): Measures how many reads or writes you can do in one second, no matter how much data you're accessing

Infrastructure as a Service (or IaaS): When a Cloud provider supplies only the bare-bones computing experience

Load balancer: Ensures that each node receives a balanced number of requests

Manual scaling: Changes are controlled by humans instead of software

Multi-cloud: A mixture of public and/or private clouds across vendors

Object storage: Storage where objects are placed and retrieved into a storage bucket

Orchestration: The automated configuration and coordination of complex IT systems and services

Persistent storage: Storage used for instances that are long lived and need to keep data across reboots and upgrades

Platform as a Service (or PaaS): When a Cloud provider offers a preconfigured platform to the customer

Private cloud: When your company owns the services and the rest of your infrastructure

Public cloud: The cloud services provided to you by a third party

Rate limits: Prevent one service from overloading the whole system

Reference images: Store the contents of a machine in a reusable format

Software as a Service (or SaaS): When a Cloud provider delivers an entire application or program to the customer

Sticky sessions: All requests from the same client always go to the same backend server

Templating: The process of capturing all of the system configuration to let us create VMs in a repeatable way

Throughput: The amount of data that you can read and write in a given amount of time

Utilization limits: Cap the total amount of a certain resource that you can provision

Docker is an easy way to package and run applications in containers. Some would consider it the most popular containerized technology. A container is a lightweight, portable, and isolated environment that facilitates the testing and deployment of new software. Within the container, the application is isolated from all other processes on the host machine. In the programming world, there is a saying that goes, “Well, it works on my machine,” meaning that a developer wrote some code that works perfectly on their local machine but does not work on others’ machines. Docker helps solve this common—and annoying—problem by providing a consistent runtime across different environments.

In this reading, you will learn more about Docker, including how to install it, and you’ll receive step-by-step instructions along the way.

Parts of Docker
The Docker ecosystem consists of the following parts:

Docker daemon. This manages running containers on a host machine called the Docker Host.

Docker CLI (command-line interface). This command-line tool interacts with Docker Daemon.

Docker Desktop. This graphical user interface (GUI) tool interacts with the daemon.

Docker Hub. This is the central repository for downloading containers.

You might hear the host machine referred to as Docker Host. Docker uses a client-server architecture as outlined by the image below. Docker supports running the client tools and daemon on different machines. This is an advantage of Docker as it allows you to manage containers on a remote server as easily as if they’re on your own workstation.

Three separate groups titled client, Docker host, and registry. Docker run, Docker build, and Docker pull are all listedunder client and have arrows pointing to Docker daemon, which is listed under Docker host. Docker daemon contains arrows pointing to Images, which are listed under registry. The Images listed under registry have an arrow pointing to the images that are listed under Docker host.
Installing Docker
Before you get started, it is recommended you read the
Getting started guide
in the official Docker documentation. Next, download and install Docker according to your operating system:

Windows.
Install Docker desktop on Windows | Docker documentation

macOS.
Install Docker desktop on Mac | Docker documentation

Linux.
Install Docker desktop on Linux | Docker documentation

When you complete the installation, you will have the Docker Daemon and Docker Desktop app installed. You are now ready to run your first container. Let’s look at how to do this on the Docker desktop app:

Open the Docker desktop app.

Select the Search bar at the top of the window.

In the search bar, type hello-world and press Enter.

The Docker desktop app opened with hello-world typed into the Search bar at the top of the window.
To the right of the found hello-world container, click Run.

The Docker desktop app opened with hello-world displayed as a search result and Run as a command option on the far rightside of the screen.
Docker downloads the image from Docker Hub and runs it. If successful, you will see a congratulatory message:

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

The Docker client contacted the Docker daemon.

The Docker daemon pulled the "hello-world" image from the Docker Hub. (amd64)

The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.

The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

https://hub.docker.com/

For more examples and ideas, visit:

https://docs.docker.com/get-started/

And that’s it! Not too bad right? Now let’s do the same thing, but this time from the command line.

Open a terminal window, type docker, and press Enter.

Note: The Docker CLI displays a summary of possible subcommands when the installation is successful.

Type docker run hello-world and press Enter.

Note: Docker displays diagnostic messages about downloading the hello-world image, then the following message if successful:

Hello from Docker!

This message shows that your installation appears to be working correctly.

Troubleshooting
You might run into a situation in which your permissions are denied while trying to connect to the Docker daemon. In this case, run the command again with sudo:

```
sudo docker run hello-world
```

If that command is successful, check your OS for a Docker group and add yourself to it. This allows you to run Docker commands without being rooted.

The Docker installation should automatically add the Docker CLI to your system path. Unfortunately, that doesn't always happen. You'll know that the Docker installation did not add the Docker CLI to your system path if you receive the message: command not found: docker. In this case, log out and back in to see if it corrects the issue. If not, consult your OS documentation on how to find the Docker binary and change your path. For additional help for troubleshooting Docker installation, view the links according to your operating system:

Linux:
Troubleshoot Docker engine installation

Window and Mac:
Workarounds for common problems

Key takeaways
When you have Docker up and running on your machine, you are in business! Docker provides so many advantages. With Docker, you can package your application and everything it needs into a portable container, and run the container! The best part is you can run the container from almost anywhere; it does not have to be on your own workstation.

Docker images
Docker images are the building blocks of Docker containers. They are lightweight, immutable, and composed of multiple layers. A Docker image contains the application code, data files, configuration files, libraries, and other dependencies needed to run an application.

In this reading, you will learn more about Docker images and their layers. You'll also learn how to build a Docker image, and you'll review an example of a Dockerfile.

Docker images and image layers
You can think of a Docker image as a template from which Docker containers are created and executed. Each Docker image is composed of multiple layers—adding or removing files from the previous layer. Each layer represents a specific set of changes made to the image and is composed based on the instructions in a Dockerfile. The instructions in a Dockerfile define how the image should be built.

Note: It's not uncommon for an image to be composed of a dozen or more layers.

The purpose of having multiple layers is to keep the final images as small as possible—you do this by reusing layers in multiple images—and to speed up the process of building containers, as Docker has to rebuild only the layers that have changed.

How to build a Docker image
The key to packaging your own application as a Docker image is to have a Dockerfile. The Dockerfile acts as your source of truth or instruction manual: It specifies how Docker should build the image and contains a series of commands to build the image. Each command builds a new layer that becomes part of the final image. A common process is to start with a base image such as Debian Linux or Python 3.10, install the libraries your application requires, then copy the application and any related files into the image. Let's take a look at a simple Dockerfile for a Python application example:

```
FROM python:3.9
```

This line of code says that you're starting from the Python 3.9 base image.

```
COPY *.py setup.cfg LICENSE README.md requirements.txt /app/

WORKDIR /app
```

This command says to copy all of the application's files to a folder inside the container named /app and make it the current working directory.

```
RUN pip install -r requirements.txt

RUN python setup.py install
```

These two lines of code run the Python commands to install the libraries required by the app. When that step is complete, build and install the app inside the container.

```
EXPOSE 8000

CMD [ "/usr/local/bin/my-application" ]
```

This command tells Docker what executable should run when the container starts and that the container will listen for network connections on port 8000.

Pro tip: To build this image from the Dockerfile, use the command: docker build. If the build is successful, Docker outputs the ID of the new image, which you can then use to start a container.

Refer to the
Dockerfile reference
 for a full list of commands that can appear in a Dockerfile.

Image names, tags, and IDs
You use tags and IDs to identify and reference Docker images. Their unique names provide a way to differentiate between specific versions of Docker images. The ID is a random string of numbers and letters, which most of the time are way too complicated to remember. But there's good news! You can assign any number of tags to the image, in addition to the ID. Tags are alphanumeric labels that help users find the correct image. Most images are tagged with the author's Github username, the name of the application, and a version number.

Pro tip: Tag the most recent version of an image with latest in addition to a version number. This makes it easy for people to find the current version of your application.

Let's look at an example:

```
csmith/my-docker-image:1.0

csmith/my-docker-image:latest

sha256:abc123def456
```

csmith is the name of the author, my-docker-image is the image name, 1.0 is the version number (and it's the latest version), and sha256:abc123def456 represents the image ID.

How to manage images
A great thing about Docker is that it caches images on a disk. Therefore, you don't need to go grab them or rebuild them every time you need them. This saves you so much time! Some of the Docker CLI (command line interface) commands you can use include:

docker image ls – This command lists the images cached locally.

docker image tag – This command applies tags to a local image.

docker image pull – This command fetches an image from a remote repository.

docker image push – This command sends a local image to a remote repository.

docker image rm – This command removes an image from the cache.

docker image prune – This command removes all unused images to reclaim disk space.

Key takeaways
Docker images—including tags and IDs—are essential for programmers to package, distribute, and deploy applications more efficiently, reducing issues and improving the stages of the software workflow. Remember, in order to have a Docker image, you must have a Dockerfile. These components work hand-in-hand; you can't have one without the other.

Using multiple containers
Imagine you are developing a web-based platform that allows users to browse products, add items to their cart, pay for items, and ship items to different addresses. This application requires multiple components to execute properly because it relies on a number of microservices. The idea behind microservices is to take a large application and break it up into smaller, more tangible, independent parts of the application that are self-contained. This allows for each part of the application to be better maintained. Because these microservices are independent of each other, you use multiple containers to test the entirety of the application to ensure everything runs smoothly. It's no surprise that in the programming world, programmers and developers work with multiple containers at a time.

In this reading, you will learn more about the use of multiple containers, commands for working with multiple containers, how related services find each other, and how to install Docker Compose and view an example.

Starting multiple containers
To start multiple containers, you need to run multiple docker run commands. A docker run command creates a container and starts it. Let's look at an example of how to create and start two containers that work together once they find each other by name.

As a programmer, you've been asked to set up a WordPress blog. You know WordPress requires a database to store its content. You create and start two containers, wordpress and db, using the following command:

```
$ docker run -d --name db --restart always \

   -v db_data:/var/lib/mysql -p 3306 -p 33060 \

   -e MYSQL_ROOT_PASSWORD=somewordpress \

   -e MYSQL_DATABASE=wordpress \

   -e MYSQL_USER=wordpress \

   -e MYSQL_PASSWORD=wordpress \

   mariadb:10
```

This command starts the mariadb database, determines a storage volume, and sets the initial password for the WordPress user. It declares two network ports open to other containers, but it is not shown on the host machine.

Now, start the WordPress container using the following command:

```
$ docker run -d --name wordpress --restart always \

   -v wp_data:/var/www/html -p 80:80 \
   -e WORDPRESS_DB_HOST=db \

   -e WORDPRESS_DB_USER=wordpress \

   -e WORDPRESS_DB_PASSWORD=wordpress \

   -e WORDPRESS_DB_NAME=wordpress \

   wordpress:latest
```

Note: The environment variable WORDPRESS_DB_HOST is set to db on the third line. This line of code is needed to refer to another container. Docker provides domain name system (DNS) services that allow containers to find each other by their name.

Networking with multiple containers
Imagine you have several customers using the same application. For security reasons, you have isolated the application and created multiple containers, one for each customer. Docker allows you to create private networks for a container or groups of containers. These private containers are able to discover each other, but no other networks will be able to find the private containers you've started. Let's look at an example: modifying the wordpress and db containers by putting them on a private network.

First, stop and delete both containers:

```
$ docker stop wordpress && docker rm wordpress

$ docker stop db && docker rm db
```

Then, create a private network for both containers to use:

```
$ docker network create myblog

0f6abeb9d85a7063298cd70082ac5e5a2f0d1624bae06619fd14dbaa0942b0e2
```

Once the containers are on private networks, restart them with the additional option -network myblog. This appears on the second to last line for both container commands.

```
$ docker run -d --name db --restart always \

   -v db_data:/var/lib/mysql -p 3306 -p 33060 \

   -e MYSQL_ROOT_PASSWORD=somewordpress \

   -e MYSQL_DATABASE=wordpress \

   -e MYSQL_USER=wordpress \

   -e MYSQL_PASSWORD=wordpress \

   --network myblog \

   mariadb:10

$ docker run -d --name wordpress --restart always \

   -v wp_data:/var/www/html -p 80:80 \

   -e WORDPRESS_DB_HOST=db \

   -e WORDPRESS_DB_USER=wordpress \

   -e WORDPRESS_DB_PASSWORD=wordpress \

   -e WORDPRESS_DB_NAME=wordpress \
   --network myblog \

   wordpress:latest
```

It’s good practice to verify that containers on other networks can’t access the private networks you created. To check this, start a new container and attempt to find the private containers you created.

```
$ docker run -it debian:latest

root@7240f1e3ddab:/# ping db.myblog

ping: db.myblog: Name or service not known
```

Docker Compose
Docker Compose is an optional tool, provided by Docker, that makes using multiple containers easy. In most instances, Docker Compose is automatically installed during the installation process of Docker Desktop. If not, follow the instructions in
Scenario two: Install the Compose plugin
 to install Docker Compose on your platform.

Docker Compose allows you to define a multiple-container setup in a single
YAML
 format, called a Compose file. (YAML is a format for configuration files that’s designed to be both human- and computer-readable.) The Compose file communicates with Docker and identifies the containers you need and how you should configure them. The containers in a Compose file are called services. Let’s look at how you can use Compose to recreate the private networks from the wordpress and db example. Run the following on your machine.

Create an empty folder and save the file below as docker-compose.yml.

```
version: '3.3'

services:

  db:

    image: mariadb:10

    volumes:

      - db_data:/var/lib/mysql

    restart: always

    environment:

      - MYSQL_ROOT_PASSWORD=somewordpress

      - MYSQL_DATABASE=wordpress

      - MYSQL_USER=wordpress

      - MYSQL_PASSWORD=wordpress

    networks:

      - myblog

    expose:

      - 3306

      - 33060
  wordpress:
    image: wordpress:latest
    volumes:
      - wp_data:/var/www/html
    ports:
      - 80:80
    networks:
      - myblog
    restart: always
    environment:
      - WORDPRESS_DB_HOST=db
      - WORDPRESS_DB_USER=wordpress
      - WORDPRESS_DB_PASSWORD=wordpress
      - WORDPRESS_DB_NAME=wordpress
volumes:
  db_data:
  wp_data:
networks:
  myblog:
```

Run the command docker compose up. This pulls up the images, creates two empty data volumes, and starts both services. The output from both services will intermingle on your screen.

The Compose file grants you control over how each service is configured, including:

Choosing the image

Setting environment variables

Mounting storage volumes

Exposing network ports

Pro tip: You can also express any option you pass to the docker run command as YAML in a Compose file.

A helpful third-party tool—that's a fan favorite of programmers—that simplifies the process of converting existing Docker run commands into Docker Compose configurations is called Composerize. Refer to
Composerize
for additional information. You can test the above Docker command in the Composerize textbox. This command defines a db service similar to the one presented above. Remember, Composerize is just a tool, and unfortunately sometimes tools come and go. It's best to understand and practice the process of converting an existing Docker run command into a Docker Compose configuration without the help of tools.

For additional information about the options you can put into a Compose file, view the
Compose file overview
documentation.

Additional Compose commands
Compose has additional commands when working with a single or multiple containers. Let's look at some examples:

docker compose pull: This fetches the latest image for each service.

docker compose up: This creates the containers and starts the service.

docker compose down: This stops the service and deletes the container.

docker compose logs: This displays the console logs from the container.

Key takeaways
Using multiple containers enables the adoption of a microservice architecture for your application. Separating a large application into smaller, independent parts allows for a more manageable approach to building, fixing, maintaining, and deploying each part of the application.

Build artifact testing
No matter what code you write, you'll need to test it. You want to create a product that is free of errors and bugs. Testing build artifacts and troubleshooting within your tests are great ways to ensure the quality of your work.

In this reading, you will learn more about different types of build artifacts, how to test a Docker container, and how to troubleshoot any issues along the way.

Build artifacts
Build artifacts are items that you create during the build process. Your main artifact is your Docker container, if you're working within a Dockerized application. All other items that you generate during the Docker image build process are also considered build artifacts. Some examples include:

Libraries

Documentation

Static files

Configuration files

Scripts

Build artifacts in Docker
Build artifacts in Docker play a crucial role in the software development and deployment lifecycle. No matter what you create with code, you need to test it. You must test your code before deployment to ensure that you catch and correct all issues, defects, and errors. This is true whether your code is built as a Docker container or built the more "classic" way. The process to execute the testing varies based on the application and the programming language it's written in.

Pro tip: It's important to check that Docker built the container itself correctly if you are testing your code with a containerized application.

There are several types of software testing that you can execute with Docker containers:

Unit tests: These are small, granular tests written by the developer to test individual functions in the code. In Docker, unit tests are run directly on your codebase before the Docker image is built, ensuring the code is working as expected before being packaged.

Integration tests: These refer to testing an application or microservice in conjunction with the other services on which it relies. In a Dockerized environment, integration tests are run after the docker image is built and the container is running, testing how different components operate together inside the Docker container.

End-to-end (E2E) tests: This type of testing simulates the behavior of a real user (e.g., by opening the browser and navigating through several pages). E2E tests are run against the fully deployed docker container, checking that the entire application stack with its various components and services functions correctly as a whole.

Performance tests: This type of testing identifies bottlenecks. Performance tests are run against the fully deployed Docker container and test various stresses and loads to ensure the application performs at expectations.

Docker makes it easy to set up and tear down tests in a repeatable and predictable way. Testing Docker containers ensures the reliability, stability, and quality of the application running within them. By testing containers, you can discover bugs and compatibility and performance issues to ensure your application functions as intended.

How to test a Docker container
Automated testing often requires supplying configuration files, data files, and test tools to the application you want to test, which unfortunately increases the size of your container. Instead, you can build a container just for testing, using your output artifact as a base image. Let’s take a look at an example:

Let’s say a Python application uses pytest as a unit testing framework and Sphix to generate documentation. You can reuse your application container and build a new image that includes the tools on top.

```
FROM myapp:latest

RUN pip install pytest pydoc

WORKDIR /opt/myapp

CMD pytest .
```

This part of the code shows that you have a container that has both the application and the test framework in it. Now it’s time for you to run the test according to the framework you chose:

```
docker run -it myapp:test
```

You can mount data files for input or configuration as a volume when you create your test container:

```
docker run -it -v ./testdata:/data myapp:test
```

What should you do if your test fails? Hopefully it won’t, but if it does, don’t worry! You can troubleshoot it. First, open the shell inside the failed container and see if you can identify the problem. If the container is still running, use the docker exec command and the container ID:

```
docker exec -it c47da2b409a1 /bin/sh
```

If you get an error running the above command, that means the container exited. You can restart the container and then try again:

```
docker start c47da2b409a1

docker exec -it c47da2b409a1 /bin/sh
```

Sometimes the container is built with automatic health checks, and Docker might terminate the container before you can investigate the issue. If this happens, you can disable the health checks in your test container by adding the command HEALTHCHECK NONE to the Dockerfile.

If you run into this type of troubleshooting often, there are tools you can add to the Dockerfile so they're always available. A couple of examples include:

jq: This is for examining JSON files.

curl, httpie, netcat: These are for testing network services.

If you want to add these tools to your test container, add it using the line below in bold:

```
FROM myapp:latest


RUN apt update && apt install -y jq curl netcat

RUN pip install pytest pydoc

WORKDIR /opt/myapp

CMD pytest .
```

Pro tip: You can never have too many automated tests. At a minimum, a good test suite will include both unit tests and integration tests.

Key takeaways
Running tests for your build artifacts and Docker containers helps ensure the reliability, stability, and quality of your work. You can never run too many tests. Tests are designed to catch bugs, identify compatibility issues, and find performance problems to assist in ensuring the application runs as intended.

Docker and GCP
Docker and Google Cloud Platform (GCP) are two types of technologies that complement each other, allowing programmers to build, deploy, and manage containerized applications in the cloud.

In this reading, you will learn more about GCP, how to run Docker containers in GCP, and how to use Cloud Run.

Google Cloud Platform
GCP is a composition of all the cloud services provided by Google. These include:

Virtual machines

Containers

Computing

Hosting

Storage

Databases

Tools

Identity management

GCP is widely used by businesses, startup companies, developers, and organizations of all sizes across a variety of industries to help their users go digital.

How to run Docker containers in GCP
You can run containers two ways in the cloud using GCP. The first way is to start a virtual machine with Docker installed on it. Use the docker run command to create a container and start it. This is the same process for running Docker on any other host machine.

The second way is to use a service called Cloud Run. This serverless platform is managed by Google and allows you to launch containers without worrying about managing the underlying infrastructure. Cloud Run is simple and automated, and it's designed to allow programmers to be more productive and move quickly.

An advantage of Cloud Run is that it allows you to deploy code written in any programming language if you can put the code into a container.

Use Cloud Run to deploy containers in GCP
Before you begin, sign into your Google account, or if you do not have one, create an account.

Open
Cloud Run
.

Click Create service to display the form.

In the form,

Select Deploy one revision from an existing container image.

Below the Container image URL text box, select Test with a sample container.

From the Region drop-down menu, select the region in which you want the service located.

Below Authentication, select Allow unauthenticated invocations.

Click Create to deploy the sample container image to Cloud Run and wait for the deployment to finish.

3. Select the displayed URL link to run the container.

Pro tip: Cloud Run helps keep costs down by only charging you for central processing unit (CPU) time while the container is running. It's unlike running Docker on a virtual machine, for which you must keep the virtual machine on at all times—running up your bill.

Key takeaways
GCP supports Docker containers and provides services to support containerized applications. Integrating GCP and Docker allows developers and programmers to build, deploy, and run containers easily while being able to focus on the application logic.

Kubernetes principles
Kubernetes is an open-sourced container orchestration platform that automates the deployment, scaling, and management of containerized applications. Kubernetes provides developers with a framework to easily run distributed systems. Kubernetes also provides developers choice and flexibility when building platforms.

In this reading, you will learn more about the principles and design philosophy behind Kubernetes, and you'll learn how declarative configuration enables the desired state reconciliation model. In addition, you'll learn about the key components of the control plane.

Kubernetes principles
Kubernetes—a cloud-native application—follows principles to ensure the containerized application runs properly. These principles take into consideration build time and run time. A container is self-contained, relying only on the Linux kernel. Once the container is built, then additional libraries can be added. In addition, containerized applications are not meant to change to different environments after they are built.

In terms of run time, each container needs to implement APIs to help the platform manage the application in the most efficient way possible. All APIs must be public, as there should be no hidden or private APIs. In addition, APIs should be declarative, meaning the programmer should be able to communicate their desired end result, allowing Kubernetes to find and implement a solution. Support is available to developers, if needed, to run applications in Kubernetes. Workloads are portable, and the control plane is able to transfer a workload to another node without disrupting the overall program.

Declarative configuration
Declarative configuration is an approach that is commonly used in Kubernetes to achieve a desired state of an application. In this approach, developers specify the desired state, but they do not explicitly define how to achieve or reach the desired state. The approach is more focused on what the desired state should be. The system will determine the most efficient and reliable way to achieve the desired state. These configuration assets are stored in a revision control system and track changes over time.

To use declarative configuration in Kubernetes, create a manifest that describes the desired state of an application. Then, the control plane will determine how to direct nodes in the cluster to achieve the desired state.

The control plane
The Kubernetes control plane is responsible for making decisions about the entire cluster and desired state and for ensuring the cluster's components work together. Components of the control plane include:

etcd

API server

Scheduler

Controller manager

Cloud controller manager

etcd is used as Kubernetes backing store for all cluster data as a distributed database. This key-value store is highly available and designed to run on multiple nodes.

The Kubernetes API server acts as the front-end for developers and other components interacting with the cluster. It is responsible for ensuring requests to the API are properly authenticated and authorized.

The scheduler is a component of the control plane where pods are assigned to run on particular nodes in the cluster.

The control manager hosts multiple Kubernetes controllers. Each controller continuously monitors the current state of the cluster and works towards achieving the desired state.

The cloud controller manager is a control plane component that embeds cloud-specific control logic. It acts as the interface between Kubernetes and a specific cloud provider, managing the cloud's resources.

Key takeaways
Kubernetes is a portable and extensible platform to assist developers with containerized applications. Kubernetes core principles and key components support developers with starting, stopping, storing, building, and managing containers.

Services
The challenge
Imagine you're developing a Python-based web application deployed in a Kubernetes cluster.

This application is composed of multiple components such as a web server, a caching layer, and a database, each running in separate Pods. These components need to communicate with each other to function properly, but there's a wrinkle: Pods have ephemeral life cycles and their IP addresses can change dynamically due to reasons like scaling, rescheduling, or node failures. But this isn't the only challenge you're facing!

Imagine that your web server, for instance, was directly communicating with the database Pod using its Pod IP address. The server would need constant updates whenever this IP changes—a manual and error-prone process.

Furthermore, consider if your caching layer is designed to handle high traffic and hence is replicated into multiple Pods for load balancing. Now, your web server needs to distribute requests among all these cache Pods. Maintaining and managing direct communication with every single cache Pod by their individual IP addresses would be a daunting task, and an inefficient use of resources.

Plus, there's the issue of service discovery. Say your web server needs to connect with a new analytics service you've just launched. It would require an updated list of all the active Pods and their IP addresses for this service—a difficult and dynamic challenge.

What is a Python developer to do in this scenario?

Services to the rescue
Fortunately, services come to the rescue in these scenarios. Services offer an abstraction layer over Pods. For starters, they provide a stable virtual IP and a DNS name for each set of related Pods (like your caching layer or database), and these remain constant regardless of the changes in the underlying Pods. So, your web server only needs to know this Service IP or DNS name, saving it from the ordeal of tracking and updating numerous changing Pod IPs.

Furthermore, Services automatically set up load balancing. When your web server sends a request to the caching layer's Service, Kubernetes ensures the request is distributed evenly among all available caching Pods. This automatic load balancing allows for efficient use of resources and improved performance.

In essence, a Service acts like a stable intermediary within the cluster. Instead of applications (like a front-end interface) directly addressing specific Pods, they communicate with the Service. The Service then ensures the request reaches the right backend Pods. This layer of abstraction streamlines intra-cluster communication, making the system more resilient and easier to manage—even as the underlying Pod configurations change dynamically.

Types of Services
Let's imagine that, with the basic challenges addressed, you've expanded your Python web application and it now includes a user interface, an API layer, a database, and an external third-party service. Different components of your application have different networking needs, and Kubernetes services, with their various types, can cater to these needs effectively.

First, you have the ClusterIP service. This is the default type and serves as the go-to choice when you need to enable communication between components within the cluster. For example, your API layer and your database might need to communicate frequently, but these exchanges are internal to your application. A ClusterIP service would give you a stable, cluster-internal IP address to facilitate this communication.

Next, you may want to expose your API layer to external clients. You could use a NodePort service for this purpose. It makes your API layer available on a specific port across all nodes in your cluster. With this setup, anyone with access to your node's IP address can communicate with your API layer by contacting the specified NodePort.

However, a NodePort might not be enough if your application is hosted in a cloud environment and you need to handle large volumes of incoming traffic. A LoadBalancer service might be a better choice in this scenario. It exposes your service using your cloud provider's load balancer, distributing incoming traffic across your nodes, which is ideal for components like your user interface that might experience heavy traffic.

Finally, you might be integrating an external third-party service into your application. Rather than expose this service directly within the cluster, you can use an ExternalName service. This gives you an alias for the external service that you can reference using a Kubernetes DNS name.

In summary, Kubernetes provides different types of services tailored to various networking requirements:

ClusterIP: Facilitates internal communication within the cluster

NodePort: Enables external access to services at a static port across nodes

LoadBalancer: Provides external access with load balancing, often used with cloud provider load balancers

ExternalName: Serves as an alias for an external service, represented with a Kubernetes DNS name

Other features

So far we've just scratched the surface of services. There are several features that extend the capabilities of services and can be employed to address specific use cases within your application's networking requirements.

Service discovery with DNS: As your application grows, new services are added and existing ones might move around as they are scheduled onto different nodes. Kubernetes has a built-in DNS service to automatically assign domain names to services. For instance, your web server could reach the database simply by using its service name (e.g., database-service.default.svc.cluster.local), rather than hard-coding IP addresses.

Headless services: Let's say you want to implement a distributed database that requires direct peer-to-peer communication. You can use a headless service for this. Unlike a standard service, a headless service doesn't provide load-balancing or a stable IP, but instead returns the IP addresses of its associated pods, enabling direct pod-to-pod communication.

Service topology: Suppose your application is deployed in a multi-region environment, and you want to minimize latency by ensuring that requests are served by the nearest pods. Service topology comes to the rescue, allowing you to preferentially route traffic based on the network topology, such as the node, zone, or region.

External Traffic Policy: If you want to preserve the client source IP for requests coming into your web server, you can set the External Traffic Policy to "Local". This routes the traffic directly to the Pods running on the node, bypassing the usual load balancing and ensuring the original client IP is preserved.

Session affinity (sticky sessions): Suppose users log into your application, and their session data is stored locally on the server pod handling the request. To maintain this session data, you could enable session affinity on your service, so that all requests from a specific user are directed to the same pod.

Service slicing: Imagine you're rolling out a new feature and want to test it with a subset of your users. Service Slicing enables you to direct traffic to different sets of pods based on custom labels, providing granular control over traffic routing for A/B testing or canary releases.

Connecting external databases: Perhaps your application relies on an external database hosted outside the Kubernetes cluster. You can create a Service with the type ExternalName to reference this database. This allows your application to access the database using a DNS name without needing to know its IP address, providing a level of indirection and increasing the flexibility of your application configuration.

Deployment
What are deployments?
Let's continue the example of the Python-based web application running in a Kubernetes cluster, specifically the web server component of the application. As traffic to your application grows, you'll need to scale the number of web server instances to keep up with demand. Also, to ensure high availability, you want to maintain multiple replicas of the web server so that if one instance fails, others can take over. This is where Kubernetes Deployments come in.

In Kubernetes, a Deployment is like your application's manager. It's responsible for keeping your application up and running smoothly, even under heavy load or during updates. It ensures your application, encapsulated in Pods, always has the desired number of instances—or "replicas"—running.

Think of a Deployment as a blueprint for your application's Pods. It contains a Pod Template Spec, defining what each Pod of your application should look like, including the container specifications, labels, and other parameters. The Deployment uses this template to create and update Pods.

A Kubernetes Deployment also manages a ReplicaSet, a lower-level resource that makes sure the specified number of identical Pods are always running. The Deployment sets the desired state, such as the number of replicas, and the ReplicaSet ensures that the current state matches the desired state. If a Pod fails or is deleted, the ReplicaSet automatically creates new ones. In other words, Deployments configure ReplicaSets, and thus, they are the recommended way to set up replication.

And by default, deployments support rolling updates and rollbacks. If you update your web server's code, for example, you can push the new version with a rolling update, gradually replacing old Pods with new ones without downtime. If something goes wrong, you can use the Deployment to rollback to a previous version.

So, in summary, a Kubernetes Deployment consists of several key components:

Desired Pod template: This is the specification that defines the desired state of the Pods managed by the Deployment. It includes details such as container images, container ports, environment variables, labels, and other configurations.

Replicas: This field specifies the desired number of identical copies of the Pod template that should be running. Kubernetes ensures this number of replicas is maintained, automatically scaling up or down as needed.

Update strategy: This defines how the Deployment handles updates. The default is a rolling update strategy, where Kubernetes performs updates by gradually replacing Pods, keeping the application available throughout the process. This strategy can be further customized with additional parameters.

Powerful features
Deployments not only help maintain high availability and scalability, but they also provide several powerful features:

Declarative updates: With a declarative update, you just specify the desired state of your application and the Deployment ensures that this state is achieved. If there are any differences between the current and desired state, Kubernetes automatically reconciles them.

Scaling: You can easily adjust the number of replicas in your Deployment to handle increased or decreased loads. For example, you might want to scale up during peak traffic times and scale down during off-peak hours.

History and revision control: Deployments keep track of changes made to the desired state, providing you with a revision history. This can be useful for debugging, auditing, and rolling back to specific versions.

A Kubernetes Deployment is typically defined using a YAML file that specifies these components. Here is an example YAML manifest.

This Deployment specifies that it should maintain three replicas of the example-container Pod template. The Pods are labeled with app: example-app, and the container runs an image tagged as example-image:latest on port 80. The default rolling update strategy will be used for any updates to this Deployment.

By utilizing Deployments, you can manage your Python web server's life cycle more efficiently, ensuring its high availability, scalability, and smooth updates.

Additional learning points
Beyond the fundamental concepts, you should be aware of a few additional features and best practices related to Kubernetes Deployments.

A fresh start: While the default update strategy is rolling updates, Kubernetes also supports a "Recreate" strategy. In the "Recreate" strategy, all existing Pods are terminated before new Pods are created. This strategy may lead to brief periods of downtime during updates but can be useful in specific scenarios where a clean restart is necessary.

Don’t get stuck: Deployments have a progressDeadlineSeconds field, which sets the maximum time (in seconds) allowed for a rolling update to make progress. If progress stalls beyond this duration, the update is considered failed. This field helps prevent deployments from getting stuck in a partially updated state. Likewise, the minReadySeconds field specifies the minimum time Kubernetes should wait after a Pod becomes ready before proceeding with the next update. This can help ensure the new Pods are fully functional and ready to handle traffic before more updates are made.

Press pause: Deployments can be paused and resumed to temporarily halt the progress of rolling updates. This feature is helpful when investigating issues or performing maintenance tasks. Pausing a Deployment prevents further updates until it is explicitly resumed.

It’s alive!: Deployments can utilize liveness and readiness probes to enhance the health management of Pods. Liveness probes determine if a Pod is still alive and running correctly, while readiness probes determine if a Pod is ready to accept traffic. These probes help Kubernetes decide whether to consider a Pod as healthy or not during rolling updates and scaling operations.

If you want to learn more about Kubernetes Deployments and their components, you can explore the provided resources below.

Key takeaways
From maintaining the desired state of your applications, managing updates and rollbacks, to ensuring high availability, Deployments provide a set of key features that streamline the deployment and management of containerized applications.

Deployments are crucial resources that manage and scale containerised applications. They automate the deployment and management of Pods and ReplicaSets.

Deployments use a declarative approach, ensuring that the application's desired state is maintained across the cluster, providing high availability. In case of Pod or node failures, the Deployment replaces the affected Pods automatically.

Deployments support rolling updates, allowing for smooth transitions during application updates with no downtime. They also offer the capability to roll back to a previous stable version in case of issues with a new update.

Deployments have several key components, including the desired Pod template (which defines the desired state of the Pods), the number of replicas (indicating the desired level of availability and scalability), and the update strategy (which defines how updates to the Pod template are handled).

Kubernetes clusters
A Kubernetes cluster comprises multiple servers (which Kubernetes calls “nodes”) that work together as a group. These nodes are virtual or physical machines that form the underlying infrastructure of the Kubernetes cluster.

Each node is capable of running containers and hosting workloads. Kubernetes clusters are designed for scalability and high availability. Nodes can be added or removed as needed as workloads vary, so applications can scale up or down seamlessly.

Nodes are interconnected and communicate with each other through the Kubernetes control plane to ensure seamless coordination and collaboration. The control plane is the brain of the Kubernetes cluster. It consists of several components that manage and monitor the cluster's overall state, including:

An API server

A controller manager

A scheduler

An etcd: This is a reliable data storage that can be accessed by the cluster of machines.

Every Kubernetes cluster has one control plane and at least one control plane node. However, multiple nodes can be tasked with running the control plane components, and these component nodes can be spread out across zones for redundancy.

The standard unit for deployment to a Kubernetes cluster is a container. Containerized applications are software applications packaged along with their dependencies, libraries, and configurations into isolated containers. Containers can be easily duplicated, ensuring easy, consistent deployment across different environments.

Kubernetes is a powerful orchestration platform designed to manage and scale containerised applications. Kubernetes automates the deployment, scaling, and management of containerised applications across the cluster's nodes. Kubernetes also manages resources across the cluster by optimally allocating CPU, memory, and storage based on application requirements. This ensures that resources are used efficiently, and it minimizes conflicts between applications. And Kubernetes also maintains the health of the cluster by employing features that automatically replace failed or unhealthy containers.

To manage a Kubernetes cluster, users specify the desired state of their applications, and then the cluster handles the actual execution and maintenance of the applications to match that desired state. This is called a “declarative approach.” The declarative approach simplifies management and reduces the need for manual intervention once the initial parameters are set.

Different types of Kubernetes clusters
Selecting the right type of cluster ensures a well-aligned Kubernetes deployment that will meet your specific business needs and objectives. Here are some of the major cluster architectures:

On-premises cluster
An on-premises Kubernetes cluster is deployed within an organization's own data center or on a private infrastructure. Deploying an on-premises cluster involves setting up the control plane and worker nodes on the organization's own hardware, and the organization is responsible for cluster maintenance. This provides complete control over the hardware, networking, and security. This is particularly suitable for situations with specific compliance or data governance requirements.

On-premises clusters are one of the primary types of Kubernetes clusters. Tools like Kubernetes kubeadm and Kubernetes Operations (kOps) are designed for deploying this type of cluster, and there are multiple custom configurations that can be used to create and manage on-premises clusters

Public cloud managed cluster
Another primary type of Kubernetes cluster is a public cloud managed cluster. Public cloud providers offer managed Kubernetes services, handling the underlying infrastructure management so it is easier for users to deploy and manage Kubernetes clusters on the cloud. This is a useful option for teams that prefer to offload cluster management tasks, but who still require the scalability and flexibility of cloud-based deployments. This type of cluster allows the organization to focus on deploying and managing applications without dealing with the complexities of cluster maintenance in the way you would with an on-premise cluster. When you run Kubernetes in the cloud, cluster maintenance is automatically handled by the cloud provider. You don't need to worry about it.

Another advantage of public cloud managed clusters is that they can be spread over zones or even regions. Just by checking a box in your configurations, you can spread your clusters geographically in case a cloud data center goes down or there are network problems. Some examples of managed services by public cloud providers are Amazon Elastic Kubernetes Service (EKS) on AWS, Google Kubernetes Engine (GKE) on Google Cloud, and Azure Kubernetes Service (AKS) on Microsoft Azure.

Private cloud managed cluster
These clusters function similarly to public cloud managed clusters, but private cloud providers manage Kubernetes services for deploying clusters within a private cloud environment. This combines the ease and flexibility of managed services with control over the private infrastructure.

Some examples of managed private cloud providers are Nutanix, OpenStack, and Hewlett Packard Enterprises (HPE).

Local development clusters
The third primary type of Kubernetes clusters are local development clusters. These are lightweight and easy-to-set-up Kubernetes environments which facilitate a fast development workflow for individual developers. These are most often set up as local development and testing clusters, and they’re primarily used for application development and testing on a developer's local machine. They are often employed during the development phase to enable rapid iteration and debugging of applications before deploying them to production clusters.

Tools commonly used to create local development clusters include Minikube, Docker Desktop (with Kubernetes enabled), and Kubernetes kind (Kubernetes in Docker). Each of these tools allows developers to spin up a single-node Kubernetes cluster locally to develop and validate applications without the need for a full-scale production cluster.

Hybrid cluster
A hybrid Kubernetes cluster coordinates on-premises and cloud environments, allowing workloads to run seamlessly across both locations. This type of cluster is suitable for scenarios in which some applications need to reside on-premises, for instance for security requirements, while others benefit from cloud scalability and services.

Edge cluster
An edge Kubernetes cluster is deployed at the edge of the network, closer to the locations of end-users or Internet of Things (IoT) devices. Edge clusters are designed to support low latency, particularly in regions or zones where power and network connectivity are scarce and expensive.

High-performance computing (HPC) cluster
HPC Kubernetes clusters are tailored for running computationally intensive workloads, such as scientific simulations or large data processing tasks. These clusters optimize performance by leveraging specialized hardware and configurations.

Multi-cluster federation
Multi-cluster federation involves managing multiple Kubernetes clusters as a single logical cluster. This allows centralized management of workloads, which are deployed across clusters similarly to the way a single Kubernetes cluster distributes workloads to multiple nodes. Multi-cluster federation facilitates global-scale deployments like disaster recovery scenarios.

Key takeaways
Kubernetes clusters are nodes which are coordinated to act as singular, cohesive units to provide a flexible and reliable platform. Clusters offer high availability, efficient scaling, and robust security. Depending on the computational purpose and requirements, you can configure a cluster to suit any organization.

Kubernetes YAML files
Suppose you're a Python developer working on a web application that's experiencing a surge in user traffic. You've containerized your application using Docker, but manually scaling the application to keep up with demand is becoming cumbersome. Kubernetes to the rescue! Kubernetes can manage and scale your containerized application automatically…if you can tell it what to do! This is where Kubernetes YAML files come in.

Kubernetes YAML files define and configure Kubernetes resources. They serve as a declarative blueprint for your application infrastructure, describing what resources should be created, what images to use, how many replicas of your service should be running, and more.

Structure of Kubernetes YAML files
Every Kubernetes YAML file follows a specific structure with key components: API version, kind, metadata, and spec. These components provide Kubernetes with everything it needs to manage your resources as desired.

apiVersion: This field indicates the version of the Kubernetes API you're using to create this particular resource.

kind: This field specifies the type of resource you want to create, such as a Pod, Deployment, or Service.

metadata: This section provides data that helps identify the resource, including the name, namespace, and labels.

spec: This is where you define the desired state for the resource, such as which container image to use, what ports to expose, and so on.

Let's illustrate this with a simple Kubernetes YAML file for creating a Deployment of your Python web application:

Key components and fields in Kubernetes YAML files
YAML files can include many other fields, depending on the type of object and your specific needs.

Pods
As you've learned, a Pod is the smallest and simplest unit in the Kubernetes object model. It represents a single instance of a running process in a cluster and can contain one or more containers. Because it is the simplest unit, a Pod's YAML file typically contains the basic key components highlighted above:

apiVersion: This is the version of the Kubernetes API you're using to create this object.

kind: This is the type of object you want to create. In this case, it's a Pod.

metadata: This includes data about the Pod, like its name and namespace.

spec: This is where you specify the desired state of the Pod, including the containers that should be running. Specifications include:

Containers: An array of container specifications, including "name", "image", "ports", and "env".

Volumes: Array of volume mounts to be attached to containers

restartPolicy: Defines the Pod's restart policy (e.g., "Always," "OnFailure," "Never")

Deployments
A Deployment is a higher-level concept that manages Pods and ReplicaSets. It allows you to describe the desired state of your application, and the Deployment controller changes the actual state to the desired state at a controlled rate. In addition to the fields mentioned above, a Deployment's YAML file includes:

spec.replicas: This is the number of Pods you want to run.

spec.selector: This is how the Deployment identifies the Pods it should manage.

spec.template: This is the template for the Pods the Deployment creates.

Services
A Service in Kubernetes is an abstraction which defines a logical set of Pods and a policy by which to access them. Key components in a Service's YAML file include:

spec.type: This defines the type of Service. Common types include ClusterIP, NodePort, and LoadBalancer.

spec.ports: This is where you define the ports the Service should expose.

spec.selector: This is how the Service identifies the Pods it should manage.

ConfigMaps
A ConfigMap is an API object used to store non-confidential data in key-value pairs. In addition to the common fields, a ConfigMap's YAML file includes the data field, which is where you define the key-value pairs.

Secrets
A Secret is similar to a ConfigMap, but is used to store sensitive information, like passwords or API keys. A Secret's YAML file includes:

type: The type of Secret. Common types include Opaque (for arbitrary user-defined data), kubernetes.io/service-account-token (for service account tokens), and others.

data: This is where you define the key-value pairs. The values must be base64-encoded.

Module 2 review
Congratulations on completing Module 2 on Docker & Kubernetes for Course 5. You've learned more about Docker and Kubernetes and about using these platforms to run applications in containers. This module walked you through the steps for installing Docker and Kubernetes to your machine—if they weren't installed already—and explored different attributes of each.

You explored Docker images and their importance in enabling programmers to package, distribute, and deploy applications efficiently. In addition, you discovered why developers use multiple containers with a large application to test the entirety of an application. Speaking of testing, you examined build artifacts and the importance of testing them to troubleshoot any issues discovered to ensure a quality product. Build artifacts are what you build during the development process and include libraries, documentation, and scripts, with the main artifact being your Docker container.

Typically, developers use GCP while working with Docker containers as it provides services to support containerized applications.

Kubernetes is another type of container you explored that provides developers with choice and flexibility when building platforms. It's designed to support developers with starting, stopping, storing, building, and managing containers. Kubernetes includes pods that are the fundamental deployment unit in a cluster, and they contain one or more containers. You learned that services offer an abstract layer over pods. In addition, services provide a stable virtual IP and DNS name for each set of related pods. You examined Kubernetes deployment and how it acts as your application's manager and is responsible for keeping your application up and running smoothly.

And lastly, you looked at how to deploy Docker containers and Kubernetes clusters to GCP. Different types of clusters include: on-premises, public and private cloud managed, local, and hybrid clusters. Kubernetes clusters are robust, flexible, and reliable units of multiple nodes that work together. You looked at the benefits of deploying and scaling Docker containers on GCP and how Kubernetes YAML files define and configure Kubernetes resources. Scaling containers allow developers to add containers as needed or allocate additional resources to each container. YAML files allow developers to manage their applications' infrastructure in a consistent and automated manner. You explored the importance of container security and actions you should take to secure your workload and network, and you also explored the actions GCP takes to secure infrastructure and operations. In addition, you learned about the best practices provided by GCP for protecting the containers you work with. Now you're ready to master the Module 2 assessment and then move on to the next module! Good luck!
Terms and definitions from Course 5, Module 2
Artifact: A byproduct of the software development process that can be accessed and used, an item produced during programming

Container registry: A storage location for container images, organized for efficient access

Container repository: A container registry that manages container images

Docker: An open-source tool used to build, deploy, run, update, and manage containers

Pod: A group of one or more containers that are scheduled and run together

Registry: A place where containers or artifacts are stored and organized

Kubernetes: An open-source platform that gives programmers the power to orchestrate containers

Puppet:

Node any system where we can run a Puppet agent
/etc/puppet/code/environments/production/manifests/site.pp - this is to create site configs

The manifests used for the production environment are located in the directory /etc/puppet/code/environments/production/manifests, which contains a site.pp file with the node definitions that will be used for this deployment. On top of that, the modules directory contains a bunch of modules that are already in use. You'll be extending the code of this deployment to add more functionality to it.

Templates are documents that combine code, data, and literal text to produce a final rendered output. The goal of a template is to manage a complicated piece of text with simple inputs.

In Puppet, you'll usually use templates to manage the content of configuration files (via the content attribute of the file resource type).

Templates are written in a templating language, which is specialized for generating text from data. Puppet supports two templating languages:

Embedded Puppet (EPP) uses Puppet expressions in special tags. It's easy for any Puppet user to read, but only works with newer Puppet versions. (≥ 4.0, or late 3.x versions with future parser enabled.)
Embedded Ruby (ERB) uses Ruby code in tags. You need to know a small bit of Ruby to read it, but it

works with all Puppet versions.

CI\CD:

GitHub is one of the most important platforms for Continuous Integration (CI) because it is the most popular site for hosting and collaborating code projects.

In order to build your project on GitHub, a CI server needs to fetch your code and other artifacts such as configuration files from source control. The exact details vary depending on the CI system in use, but the basic outline is this:

1. Log into GitHub and go to the personal access tokens (classic) page.
2. From the **Generate new token** drop-down menu, select **Generate new token (classic)**.
3. Give the token a name and select the permissions your CI system needs (these will usually be listed in the CI system documentation).
4. Click **Generate token**. Make sure you save the token. It will only be displayed once.
5. Provide your GitHub username and the token to your CI system.

After you have integrated the two, the CI system will watch your GitHub project for new code commits. Usually this means watching for three types of events:

6. Commits to the master branch.
7. Commits to an alternative branch.
8. Pull requests being merged into master or another branch.

These events can trigger the CI system to fetch the latest code and run the CI pipeline in that branch.

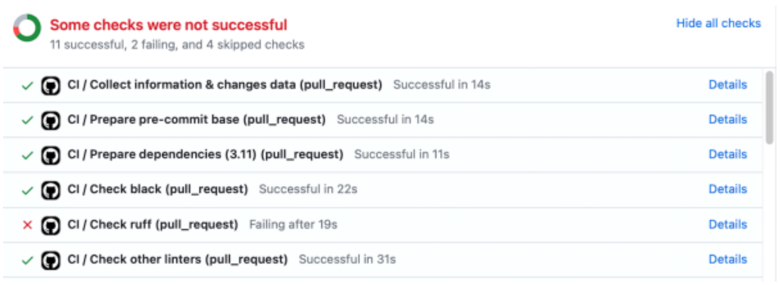
Webhooks

GitHub is able to notify CI tools about code changes through webhooks. A webhook is a URL provided to GitHub by the CI system. GitHub will send an HTTP request to the webhook URL whenever certain events happen in the GitHub repository (such as the three scenarios outlined above). GitHub will pass certain information in the HTTP request that informs the CI system of the commit.

Usually the CI system will use your GitHub API token to add the webhooks to your GitHub project for you. In some cases, they will provide instructions and a webhook URL that you can paste into GitHub yourself.

GitHub's status checks

GitHub's status checks notify you if your commits meet the conditions set for the repository to which you're contributing. A status check is a GitHub API feature that allows an external CI system to mark a pipeline as pending, succeeded, or failed. A screenshot below shows a display of status checks:



Some checks were not successful

11 successful, 2 failing, and 4 skipped checks

Hide all checks

✓ CI / Collect information & changes data (pull_request) Successful in 14s Details

✓ CI / Prepare pre-commit base (pull_request) Successful in 14s Details

✓ CI / Prepare dependencies (3.11) (pull_request) Successful in 11s Details

✓ CI / Check black (pull_request) Successful in 22s Details

✗ CI / Check ruff (pull_request) Failing after 19s Details

✓ CI / Check other linters (pull_request) Successful in 31s Details

A single pipeline may create several status checks in GitHub. For example:

- Build - succeeded
- Unit tests - succeeded
- Integration tests - pending
- Security scan - pending

GitHub will display the status checks on the appropriate commit or branch, so that you can watch the status of the pipeline without having to find it in the CI tool. For an example of what status checks look like, see About status checks.

Key takeaways

CI with GitHub benefits developers and the community. After you have integrated with GitHub, the CI system will watch your GitHub project for new code commits. GitHub communicates changes with the CI system through webhooks. You can review the status checks to see if your commits meet the conditions.

From <https://www.coursera.org/learn/configuration-management-cloud/supplement/cQzJQ/integration-with-github>

CI best practices

Continuous integration (CI) is a software development practice in which code changes occur automatically, frequently, and safely to integrate into a shared repository. With each integration, an automated build is triggered and tested to determine and resolve integration issues early.

Principles and benefits

Key principles of CI include:

- Integration
- Builds
- Tests
- Feedback
- Version control

CI allows for frequent integration, as developers commit their changes to a shared repository daily. It also ensures the changes are tested and issues are resolved. CI relies on automation to build the application code and execute tests to manage integrated changes and to ensure consistent results. Because CI is able to detect integration issues early, this allows for CI to provide quick feedback to developers, alerting them of issues and providing time to allow them to correct any problems. Lastly, running the CI tests on every pull request will highlight which changes caused the tests to fail, guaranteeing a reliable history of the codebase for developers.

Let’s look at an example of where you’d see this in the real world: Imagine that a team working on a modern microservice-based application has been suffering and exhausted due to subtle errors showing up in production that should have been caught earlier. The team makes the decision to invest time and resources into building a strong set of integration tests.

After the unit tests run successfully, the team prepares the CI server to deploy all the microservices into a Kubernetes cluster and run the automated integrated test suite. Each microservice developer writes tests to check the interactions between their service and the other services it depends on. If any of those tests fail, the pull request is not merged and the code is not deployed to production.

Core practices

CI is a way to develop software to make it easier, faster, and more reliable. CI is composed of three core practices, which include:

- Automated building
- Automated testing
- Version control system integration

Automated builds involve utilizing tools and scripts to automatically compile the source code into executable binaries or artifacts. Automated testing involves running different types of tests automatically to verify the stability of code changes made by developers, providing fast feedback to developers on any issues. Version control system integration allows a way to manage and track code changes efficiently and effectively.

Key takeaways

Continuous integration enables faster feedback, higher quality software, and a lower risk of bugs and conflicts in your code. CI is a way for developers to ensure that their code is always up to date and ready to deploy. CI ensures that reliable software is getting into the hands of users.

From <https://www.coursera.org/learn/configuration-management-cloud/supplement/JZNOM/ci-best-practices>

CI testing

You’ve learned that continuous testing means running automated test suites every time a change is committed to the source code repository. In practice, this usually means running the tests as part of a CI/CD pipeline, in between the build and deploy stages. In this reading, you’ll learn about continuous integration (CI) testing: why it’s important, what tools you can use, and how to test different types of applications, like the web, a database, or a microservice.

Integration testing

Continuous integration is when developers integrate code into a shared repository frequently. Benefits of continuous integration include revision control, build automation, and automated testing which allows you to detect errors quickly and locate them more easily. Integration tests are tests you conduct to make sure different parts of the application work together, like modules or services, as opposed to individual units of code. They're one type of test that is typically run during the CI pipeline. The purpose of integration testing is to make sure that any recent changes you or your team has made haven't broken other parts of the system and to verify that everything is working together as expected. A few benefits of CI testing include repeatability of your testing, continuous integration and testing, the ability to run builds or tests in parallel with other team members, and rapid feedback.

There are different types of CI tests that you can perform in the CI pipeline:

- Code quality tests are used to check the quality and complexity of your code(s) and identify if there are any code defects.
- Unit tests which are used to test an individual unit within your code, like a function, module, or set of processes. Unit testing checks that everything is working as expected.
- Integration tests are used to verify that the different modules or parts of your application are working together.
- Security or license tests are used to make sure that your software or application is free from threats, vulnerabilities, and risks. This allows developers to identify if and where there are security risks so they can be fixed earlier rather than later.

You've learned the different kinds of CI tests you can perform, but what about the tools you can use to run the tests?

- If you want to test the integrations between web services, you can use a programming framework like PyTest or its equivalent (if the app you're using is Python).
- If you want to test a browser-based app or site, you can use a framework like Selenium or Playwright to load the web pages and test functionality.

Refer to Continuous testing and continuous improvement for more information on the tools you can use to run integration tests.

After you've chosen the CI tool that fits your needs, it can run the designated tests and halt the deployment of your software or application if some or any of the tests fail. There are also "code coverage" tools that will scan your code for you and tell you if it appears you've missed anything in your test suite.

End-to-end testing

Integration testing verifies that the components of your application are working together as intended and usually occurs in the CI pipeline. End-to-end (E2E) testing is similar. Although it occurs in the CD pipeline, it is used to test the functionality and performance of your entire application from start to finish by simulating a real user scenario. The objective of E2E testing is to identify any errors or bugs that appear when all of your components have been integrated, so you know if the entire application operates as it should. You can gain insights into how end users are experiencing your application, giving you a better understanding of the quality of your product before it's deployed.

Key takeaways

Continuous integration testing is used to test different components of your application throughout the CI pipeline. It allows you to verify that any code changes made by developers working independently haven't created errors or broken other parts of the system. Continuous integration testing can also lead to continuous testing, allowing you to automate different processes along the CI/CD pipeline.

It's important to note that if you're going to automate the entire process of deploying your code to production, you'll want to significantly reduce the risk that the changes you make will break the production system. It will be up to you and your team to weigh the risk/reward ratio of whether you want to perform less testing and quickly make changes to the production system or perform more testing and delay changes to the production system. There is no "one-size-fits-all" solution for all systems, so choose the tests and the tools that make the most sense for your project.

From <https://www.coursera.org/learn/configuration-management-cloud/supplement/hQ5Zx/ci-testing>

https://www.lucidchart.com/blog/value-stream-mapping-for-devops

Github and delivery

GitHub can facilitate your efforts in Continuous Integration and Continuous Delivery (CI/CD). Learning about how to utilize this repository will benefit you down the road.

How GitHub supports CI/CD

GitHub supports external CI/CD tools by providing webhooks and APIs that allow those tools to become part of the pull request process. For example, GitHub can be set up to refuse to merge a pull request until several actions are completed:

- The PR is reviewed and signed off by one or more code reviewers.
- The CI build process completes successfully.

- The CI test suites run successfully.
- The PR submitter has acknowledged the project's license, standards, and/or code of conduct.

The feature of GitHub that automates CI/CD is called GitHub Actions.

GitHub Actions

GitHub Actions is a feature of GitHub that allows you to run tasks whenever certain events occur in your code repository. With GitHub Actions, you are able to trigger any part of a CI/CD pipeline off any webhook on GitHub.

To get started with GitHub Actions, go to the **Actions** tab in the GitHub repository. When opening the tab for the first time, you'll find a description of GitHub Actions and some suggested workflows for your repository. These workflows build the code in your repository and run your tests either on GitHub-hosted virtual machines or on your own machines. You'll receive the results of the tests in the pull request.

GitHub Actions has more than 13,000 pre-written and tested CI/CD workflows. If you would like to write your own workflows or customize an existing one, you can do it using YAML files.

Because GitHub Actions is a general-purpose engine that runs tasks in response to events in the repository, you can use it for more than just CI/CD pipelines. Some examples:

- Running a nightly build in the master branch for testing.
- Cleaning up old issues or bug reports.
- Creating a full-fledged bot that responds to comments or commands on a PR or issue. ("Issues" are like bug reports or support tickets.)

Key takeaways

GitHub Actions supports your CI/CD. It offers workflows that you can use right away or customize for your own repository.

From <https://www.coursera.org/learn/configuration-management-cloud/supplement/oUd1s/github-and-delivery>

Tools for end-to-end tests

You can use testing frameworks such as Selenium and Puppeteer to conduct an E2E testing. Most of these UI testing frameworks target web applications and they tend to be JavaScript-based as web apps almost always use HTML+JavaScript (react.js, angular, vue.js) for their user interface components. For Python developers, when you utilize these frameworks, you will need to use Flask or Django. Flask is commonly used to build web apps and it can handle microservices. Django is used for larger or more complex applications.

Here are some of the testing tools you can use with Flask and Django:

- Selenium is a popular open-source tool that allows for automated testing across various platforms and browsers. It supports multiple programming languages like Java, C#, and Python.
- Puppeteer is a Node library that provides a high-level API to control Chrome or Chromium over the DevTools Protocol. It can be used for testing Chrome extensions and for generating screenshots and PDFs of pages.
- Cypress is a JavaScript-based testing framework that doesn't use Selenium. It allows you to write all types of tests: E2E tests, integration tests, and unit tests. However, Cypress doesn't support Python.
- Appium is an open-source tool for automating native, mobile web, and hybrid applications on iOS mobile, Android mobile, and Windows desktop platforms. It's widely used for end-to-end testing of mobile applications.
- Protractor is an end-to-end test framework for Angular and AngularJS applications. It runs tests against your application running in a real browser, interacting with it as a user would.

From <https://www.coursera.org/learn/configuration-management-cloud/supplement/IPI9O/end-to-end-tests>

For this lab you created 2 Git repositories:

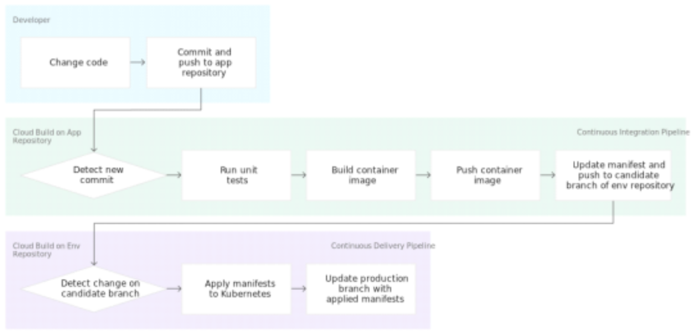
- app repository: contained the source code of the application itself
- env repository: contained the manifests for the Kubernetes Deployment

When you pushed a change to the **app repository**, the Cloud Build pipeline ran tests, built a container image, and pushed it to Artifact Registry. After pushing the image, Cloud Build updated the Deployment manifest and pushed it to the **env repository**. That triggered another Cloud Build pipeline that applied the manifest to the GKE cluster and, if successful, stored the manifest in another branch of the **env repository**.

The app and env repositories are kept separate because they have different lifecycles and uses. The main users of the **app repository** are actual humans and this repository is dedicated to a specific application. The main users of the **env repository** are automated systems (such as Cloud Build), and this repository might be shared by several applications. The **env repository** can have several branches that each map to a specific environment (you only use production in this lab) and reference a specific container image, whereas the **app repository** does not.

When you finished this lab, you created a system where you can easily:

- Distinguish between failed and successful deployments by looking at the Cloud Build history.
- Access the manifest currently used by looking at the production branch of the **env repository**.
- Rollback to any previous version by re-executing the corresponding Cloud Build build.


Developer
Change code
Commit and push to app repository
Cloud Build on App Repository
Continuous Integration Pipeline
Detect new commit
Run unit tests
Build container image
Push container image
Update manifest and push to candidate branch of env repository
Cloud Build on Env Repository
Continuous Delivery Pipeline
Detect change on candidate branch
Apply manifests to Kubernetes
Update production branch with applied manifests

This exemplar is a walkthrough of the previous Qwiklab activity, including detailed instructions and solutions. You may use this exemplar if you were unable to complete the lab and/or you need extra guidance in competing lab tasks. You may also refer to this exemplar to prepare for the graded quiz in this module.

From <https://www.coursera.org/learn/configuration-management-cloud/supplement/ZlfUS/exemplar-cicd-pipeline-using-gcp>

Module review:

Module 4 review

Congratulations on completing the last module for *Course 5: Configuration Management and the Cloud*. You're making great progress! You've learned so much about the continuous integration and continuous delivery process, which ensures your software functions as expected and any changes made are tested properly.

You've discovered what a CICD pipeline is and the importance of automating the tools to build, test, package, and deploy an application when code changes are integrated. You learned that DevOps tools are the software tools to help you accomplish the automated process throughout the DevOps lifecycle. The tool you use will be dependent on the stage of the lifecycle you are in. Implementing containers in your CI/CD processes can improve collaboration amongst developers and ensure the delivery of efficient and reliable applications. It's important to run continuous testing as part of a CI/CD pipeline and, depending on your test results, to implement the changes necessary to continually improve efficiency and reduce errors.

In addition, you learned about the continuous integration platform, Github, and the benefits it provides to the coding community. You also explored Cloud Build—a CI/CD service—provided by GCP, its core components, and the benefits of using Cloud Build to automate building, testing, and deploying code changes to various environments. You examined CI best practices including automated building, testing, and version control system integration. And you looked at CI testing to detect and correct bugs or defects in your code to ensure your codebase stays stable and manageable.

You acquired skills in continuous delivery and the iterative development cycle. You learned that CD is essential with CI and automated testing and that the use of feature flags, increment rollout, and blue-green deployments will minimize problems that are commonly caused by human error. In addition, you learned that Github supports CI/CD tools by providing webhooks and APIs and allows those tools to become part of the pull request process. You looked at how value stream mapping optimizes the flow of materials and information required to successfully deliver a product to clients and how configuration management provides a solution to simplify and effectively manage

changes to code.
Lastly, you looked at end-to-end tests and how they're a comprehensive evaluation of an application that mimic real-world use. You explored the four different types of releases and the management that must be used to ensure an effective release. You also now know that if a software incident occurs, you can conduct a post-mortem, or incident analysis, to determine what happened, how it happened, and how to create a solution to prevent disruptions in the future.

From <https://www.coursera.org/learn/configuration-management-cloud/supplement/aMXIb/module-4-review>

Glossary terms from course 5, module 4

Terms and definitions from Course 5, Module 4

Continuous delivery: Any changes to the software are tested and then deployed to users and servers as soon as they are verified
Continuous deployment: Automates the deployment of code to production
Continuous integration: Constantly adding updates and improvements to software
DevOps: Describes the steps of the software development lifecycle beyond writing code, the union between the development team and the operations team
DevSecOps: Adding security testing and protection to the software development lifecycle
Production: The software is pushed out to the end users from a cloud server
Staging: A strategic DevOps approach where we specify the build steps and tests

From <https://www.coursera.org/learn/configuration-management-cloud/supplement/LkB2a/glossary-terms-from-course-5-module-4>

MOD 6:

NALSD

NALSD, also known as non-abstract large system design, is a discipline and a process invented by Google that aims to give SREs (site reliability engineers) the ability to assess, design, and evaluate large systems. NALSD is the process of designing complex and substantial systems, such as software applications, hardware systems, or even organizational structures, with a focus on practical, concrete details rather than on abstract or theoretical concepts. NALSD emphasizes the tangible and real-world aspects of system design and implementation.
NALSD combines elements of capacity planning, component isolation, and graceful system degradation that are crucial to highly available production systems.

Key characteristics of NALSD

NALSD requires DevOps teams to think about scale and resilience during the design process. It separates the design process into two phases. The two phases have the following key characteristics:

Phase 1: Technical design

Phase 1 is an iterative process that involves multiple rounds of design and refinement. It's common to create prototypes, conduct feasibility studies, and gather feedback from stakeholders to continuously refine the technical design. In this phase, the team tries to answer two questions about the proposed design:

- Is it possible? Will the design even work?
- Can we do better? Can we make it faster, simpler, or cheaper?

Phase 2: Scaling up

In Phase 2, the team assesses whether the system design is feasible at scale. They consider how the system will perform when subjected to significant increases in load. Scalability is essential to ensure that the system can accommodate growth without a dramatic loss of performance. What if you suddenly add a million users to the system? How will the system be able to accommodate a random increase in users?

- Is it feasible? Will it work at scale? Is it cost-effective?
- Is it resilient? What happens if the database goes down?
- Can we do better? Are there changes or additions that we need to make?

Three key goals of NALSD

Three of the key goals of NALSD are the following:
The first is proper capacity planning. Capacity planning is understanding how to properly size each component and how to properly plan for growth. This goal involves careful monitoring, performance analysis, and prediction of growth trends to prevent resource exhaustion or over-provisioning. Capacity planning is crucial in NALSD design because it involves estimating the

required resources (CPU, memory, storage, network bandwidth, etc.) to meet current and future demands.
The second is component isolation. In NALSD design, a fundamental principle is component isolation, highlighting the importance of designing each element of the system to maximize simplicity, modularity, and independence from one another. The "do one thing and do it well" philosophy encourages developers to create components that have a clear and specific purpose.
The final goal is graceful degradation. This is the idea that parts of the system should continue to work when another part fails, rather than everything failing at once. For example, in a web application, if a database server becomes unavailable, the system might switch to a read-only mode, allowing users to access existing data while blocking new updates until the database is restored.

Google's NALSD Workbook

The NALSD Workbook was created by Google's SRE team. It is designed to help engineers and developers with the design and architecture of large-scale, reliable systems.
The NALSD Workbook contains valuable insights, best practices, and guidelines for designing and building complex and scalable systems that can handle high loads and remain reliable. Engineers often refer to such resources to improve their system design skills and create more robust and efficient software and infrastructure. If you're interested in learning more about large-scale system design, this workbook is a fantastic resource and can be found here.

Key takeaways

Here are three key takeaways from this reading on NALSD:

- **Definition of NALSD:** NALSD, or Non-Abstract Large System Design, is a discipline and process introduced by Google, primarily aimed at empowering site reliability engineers (SREs) to assess, design, and evaluate large-scale systems.
- **Two phases of NALSD:** Phase 1 involves continuous refinement through feedback, prototyping, and feasibility studies. It seeks to answer: "Is it possible?" and "Can we do better?" Phase 2 evaluates the system's feasibility and resilience at scale, considering how it will perform under significant load increases.
- **Three key goals of NALSD:** Proper capacity planning, component isolation, and graceful degradation are the three goals of NALSD. These goals are in place so that the system can continue functioning even when individual parts fail.

As previously mentioned, if you are interested in learning more about NALSD, review Google's Non-Abstract Large System Design Workbook.
Mark as completed
Like
Dislike
Report an issue

From <https://www.coursera.org/learn/automating-real-world-tasks-python/supplement/sZaJD/nalsd>

Apis:

RESTful APIs

RESTful APIs were originally conceptualized by Roy Thomas Fielding in his 2000 PhD thesis. Unlike APIs which directly open up ports to the entire internet and directly connect, RESTful APIs rely on the HTTP protocol. The HTTP protocol, in turn, can be further secured using HTTPS, and API endpoints can authenticate users via authorization tokens, API keys, or other security mechanisms. RESTful APIs use HTTP requests to perform CRUD (create, read, update, delete) operations on resources.

RESTful methods

RESTful APIs work by associating methods (functions) with resources. Some of the most commonly used HTTP request methods are:

- GET: The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.
- HEAD: The HEAD method asks for a response identical to a GET request, but without the response body.
- POST: The POST method submits an entity to the specified resource, often causing a change in state or side effects on the server.
- PUT: The PUT method replaces all current representations of the target resource with the request payload.

You can use GET to obtain information (called a "request") from a RESTful API endpoint, which would deliver a replay from the endpoint (called a "response"). Each type of response has a three-digit code associated with it; these are HTTP response codes. You might already be familiar with a 404 (not found) response. If a request succeeds, the response status code is 200, and the response will contain a message payload, typically JSON (JavaScript Object Notation). It is important to note that RESTful APIs almost always use JSON, but RESTful APIs can also be used to send and receive files, as well as stream data.

JSON

JSON is a data-interchange format used in RESTful APIs to facilitate communication between clients and servers. In RESTful services, JSON serves as the standard payload format for transmitting data. When a client makes a request, the server processes it and sends back a response, often in JSON format. This structured format allows for easy parsing, ensuring both the server and client can interpret the data consistently. With its key-value pairs, JSON is both human readable and machine friendly, making it a popular choice for web-based APIs.

Additional protection

Another important thing to note is that RESTful APIs provide a layer of protection over existing cloud assets, such as a database. Instead of allowing the entire internet to query your database, you can put an API in front of it and allow the API to serve as an intermediary: an endpoint associated with that resource. You can then force authentication using a JavaScript Web Token (JWT) or a third-party authentication provider. This layer of protection not only provides security, but it also separates control logic from data, so that rate-limiting, usage analytics, and caching can be added on top to improve quality of service.

Compatibility

Lastly, RESTful APIs are callable from any programming language that can support HTTP or HTTPS. This includes Python but also C#, JavaScript, Swift, Go, and many others. RESTful APIs, relying on only HTTP, allow any modern computers to talk to each other.

Key takeaways

RESTful APIs are a fundamental and versatile part of web development. Knowing how to design, consume, and work with them is essential for building modern web applications, ensuring they can communicate effectively with other services.

From <https://www.coursera.org/learn/automating-real-world-tasks-python/supplement/eOX83/restful-apis>

What is REST architecture?

REST stands for Representational State Transfer. REST architecture is an architectural style for designing networked applications and web services. It was invented as a standard way for clients and servers to communicate with each other over the internet.

REST is designed to be stateless, meaning the server does not have to remember any information about the client between requests. Every request carries all the parameters and data needed for the server to satisfy that request.

Constraints with the REST architecture

The six constraints or principles of REST each serve a specific purpose in guiding the design of RESTful systems, contributing to the overall scalability, simplicity, and interoperability of the architecture. The six constraints are:

- **Uniform interface** - There should be consistent methods for clients to access and change resources on the server using standard HTTP (Hypertext Transfer Protocol) conventions.
- **Stateless** - Every piece of information the server requires to process the request should be within the request. There shouldn't be any leftover information on the server between requests.
- **Cacheable** - Every server response should indicate whether the data can be cached on the client and the length of time is needed to cache the data.
- **Client-server** - The client and server can evolve independently. The REST interface serves as a “contract” between them.
- **Layered system** - An application should be split into layers. Each layer of the application handles a particular concern (data access, business logic, presentation, etc) and acts independently from the other layers.
- **Code on demand (optional)** - Servers can also provide code to be executed on the client. This enables the client to change its behavior dynamically.

HTTP protocol with REST

REST is also designed to run on top of HTTP. This design enables clients and servers to communicate over the public internet using standard HTTP conventions. Companies will often publish their REST API (Application Programing Interface) so that developers can make use of it. Nearly any programming language is capable of speaking HTTP, so you can use your favorite language, like Python, to make REST API calls.

All interaction between client and server takes place over HTTP, using standard HTTP features: verbs, headers, and data payloads. Almost everything the server needs to know is included in the request URL itself.

For example, a photo-sharing app might send a series of HTTP requests to a REST API server that look like the following (command is listed first and then the action the command is performing is listed after the dash):

- **GET /api/v1/albums** - get the list of photo albums
- **GET /api/v1/albums/1234/pictures** - get the list of pictures in album 1234
- **GET /api/v1/pictures/5678** - get the details for picture 5678
- **GET /api/v1/pictures/5678/comments** - get the comments for picture 5678

HTTP allows clients to GET, PUT, and DELETE resources. Clients can also POST queries with complex data, such as performing a search or transferring money between accounts. The PATCH verb allows clients to update a resource by just sending what has changed.
REST APIs often allow the client to adjust their behavior by sending additional headers with the request. Headers might include authentication, enable optional features, or allow the client to request that the server send data in specific formats (e.g. JavaScript Object Notation, JSON, or eXtensible Markup Language).
The client may send data in the body of its request, and the server replies with data in the response body. The format of the data is controlled by a header (see above).

What is the Richardson Maturity Model?

The Richardson Maturity Model, also known as RMM, is a framework that categorizes and describes different levels of implementation for RESTful APIs based on their adherence to the six constraints referenced earlier in this reading. RMM is a way of assessing the sophistication of a REST API based on how compliant it is with the REST constraints.
The Richardson Maturity Model consists of four levels, each representing a progressive level of adherence to the principles of REST:

- **Level 0** - A single URI (uniform resource identifier) and a single verb (usually GET or POST)
- **Level 1** - Multiple URIs but still a single verb
- **Level 2** - Makes use of URIs and multiple methods, but is not HATEOAS (Hypermedia as the Engine of Application State)
- **Level 3** - Full HATEOAS

HATEOAS indicates that the server's responses should include hyperlinks for the client to access related resources. For example, in the picture gallery app example above, the GET request for albums should return a list of albums. Each album should include its name, ID, and links to retrieve album details, comments, and pictures. With a full Level 3 REST implementation, the client would not need to hardcode URIs of the resources it needs. The URIs would be discoverable from the server's responses.
To check out some further information with how to create REST APIs in Python, check out this link here. If you're interested in further information for REST APIs with GCP, click on this link here.

Key Takeaways

REST architecture is based on six key constraints, including a uniform interface for consistent interactions, statelessness for efficient communication, cacheability for improved performance, separation of client and server concerns, layered system organization, and the optional ability for servers to provide executable code to clients. REST APIs are often designed to run on top of the HTTP protocol, utilizing standard HTTP features for communication. APIs are difficult to change after they are published and being utilized. Invest the time to create clean, rational, extensible APIs right from the start.
Mark as completed
Like
Dislike
Report an issue

From <https://www.coursera.org/learn/automating-real-world-tasks-python/supplement/ymF9A/what-is-rest-architecture>

Logging

Many Python programmers develop programs that require them to observe values to see what is going on by using **print()** statements. **print()** is a built-in Python function used for simple, immediate output to the console which can be used for debugging or quick information display. Logging is a more sophisticated and configurable logging framework that allows developers to control the level of detail, destination, and formatting of log messages. Logging makes it suitable for production-grade applications to track and analyze program behavior. **print()** statements are great for development, but for production, each one of these function calls increases overhead.
One example of **print()** statements causing negative overhead is if low-priority problems arise in a script. A third-party API endpoint isn't responding in time, so the application tries again and isn't able to get a response. In these cases, the system doesn't tell you that the system is broken.
For more severe problems, such as a totally offline server dependency or repeated denials for incorrect passwords, you want to know immediately. For your logs, you want to filter out errors that may go away due to external factors (data about outages and potential breaches).
When logging problems occur, they generally do not happen in isolation. Having a massive amount of noise inside log files can make it hard to trace problems. This ideology is pointed out

in the “Observing Application Behavior” chapter of Michal Jaworski’s book, *Expert Python Programming, 4th edition*.
You might want to set up a way to notify someone in case problems arise. Filtering problems by severity level is a fantastic place to start. Logging allows this filtering with the levels: **DEBUG**, **INFO**, **WARNING**, **ERROR**, and **CRITICAL**.

Printing vs. logging

Printing and logging are both methods you can use to display information, but they serve slightly different purposes and have different implications, depending on the context.

Printing

print() is a function that prints objects (strings, but also most anything else). The syntax for using the **print()** function to print to a file is:

```
print(open(“my_file.txt”,”r”).read(),file=open(“new_file.txt”,”w”))
```

print() objects sent to the text stream file must be provided as keyword arguments if they are separated by sep. These statements might also be followed by end. sep, end, file, and flush.
The stream file is generally what you interact with, but sometimes you will not use Python with console output, especially in a cloud context where code is deployed that runs on some other computer. In most cases, anything printed out inside the Python code will end up in logs. This is because logs capture all or nothing regarding **print()**.

### Logging

The logging module provides an object-oriented way to print out statements with the additional advantage of using severity levels. More advanced features include the ability to filter which messages you care about, to route to more destinations than just the console/terminal, and to see the difference between normal **print()** statements and **print()** statements used for debugging purposes.

Built-in logging levels

The logging module provides five built-in log levels. In order of severity, they are **debug**, **info**, **warning**, **error**, and **critical**. These are referred to as methods of the logging object.
The output of a logging statement will resemble:

```
[LOG_LEVEL]:root:[MESSAGE]
```

The name “root” in this context means logging was called directly instead of setting up a logger by name.
The following is an example of import logging:

```
# Set up basic configuration to display all log levels
logging.basicConfig(level=logging.DEBUG)
# Log messages for each severity level
logging.debug("This is a DEBUG message.")
logging.info("This is an INFO message.")
logging.warning("This is a WARNING message.")
logging.error("This is an ERROR message.")
logging.critical("This is a CRITICAL message.")
```

This prints out:

```
DEBUG:root:This is a DEBUG message.
INFO:root:This is an INFO message.
WARNING:root:This is a WARNING message.
ERROR:root:This is an ERROR message.
CRITICAL:root:This is a CRITICAL message.
```

This print out includes all of the pre-defined logging points.

Handlers

Handlers are advanced ways to manage and route logs. Handlers define the output destinations for log messages, allowing you to control where the log data goes, such as writing to files, sending emails, or printing to the console.
The most basic handler is **StreamHandler**. The **StreamHandler** class allows you to configure logging to display log messages on the screen, making it useful for providing feedback, debugging, and monitoring during the execution of a program. This is the **print()** equivalent for logging, but **StreamHandler** also allows routing from arbitrary objects. Handlers can be attached to user-defined loggers using **addHandler()** or can be directly invoked.
Here is an example of a custom logger with a **debug** and a **StreamHandler** attached to it with a different severity level:

```
import logging
# Setting up the logger and StreamHandler
# Creating a logger
stream_logger = logging.getLogger('stream_logger')
stream_logger.setLevel(logging.DEBUG)  # Set logger to capture all messages from DEBUG level
and above
# Ensure no previous handlers are attached
stream_logger.handlers = []
# Creating a StreamHandler
stream_handler = logging.StreamHandler()
stream_handler.setLevel(logging.INFO)  # Set handler to display only messages from INFO level
and above
# Adding handler to logger
stream_logger.addHandler(stream_handler)
```

```
# Logging messages at different levels
stream_logger.debug("This is a DEBUG message for stream_logger.")
stream_logger.info("This is an INFO message for stream_logger.")
stream_logger.warning("This is a WARNING message for stream_logger.")
stream_logger.error("This is an ERROR message for stream_logger.")
stream_logger.critical("This is a CRITICAL message for stream_logger.")
```

output:

```
This is an INFO message for stream_logger.
This is a WARNING message for stream_logger.
This is an ERROR message for stream_logger.
This is a CRITICAL message for stream_logger.
```

Different types of logging handlers

Besides StreamHandler, there are also many other handlers:

- FileHandler
- NullHandler
- WatchedFileHandler
- BaseRotatingHandler
- RotatingFileHandler
- TimedRotatingFileHandler
- SocketHandler
- DatagramHandler
- SysLogHandler
- NTEventLogHandler
- SMTPHandler
- MemoryHandler
- HTTPHandler
- QueueHandler
- QueueListener

For more information about logging handlers, click here.

Logging business data to a database

An advanced use case, but one very common in most cloud operations, is logging business-relevant data, such as logins and expensive requests, to a database. This logging would also generate alerts for outages and other items that require immediate attention. To do this with Python logging, one may combine the capabilities of the logging library with a database library. This enables the creation of custom handlers or loggers that interact with the database, allowing seamless storage and retrieval of crucial operational data to enable effective monitoring.

Key takeaways

These takeaways emphasize the significance of logging for effective software development and production monitoring, as well as the distinction between using `print()` statements and the more structured approach of logging with severity levels and handlers.

- **Overhead of `print()` statements in production:** Although using `print()` statements is convenient for development, it can lead to increased overhead in production environments.
- **Distinguishing severity levels for effective monitoring:** Differentiating between various issues' severity levels is crucial. By using severity levels in logging, you can effectively filter out transient errors and monitor significant problems.
- **Using logging handlers for advanced log management:** Logging handlers are essential components for directing log messages to various destinations. They provide advanced capabilities, such as filtering messages, routing to multiple destinations, and enabling custom loggers.

Mark as completed

Like

Dislike

Report an issue

From <https://www.coursera.org/learn/automating-real-world-tasks-python/supplement/CUx3q/logging>

The most common types of exceptions

When an error arises while a program is being executed, one of Python's built-in exceptions may occur. Some of the most common exception types in Python are the following:

- `NameError` - usually due to a typo in a variable name
- `AttributeError` - also usually a typo, in calling a method on an object
- `ValueError` - parameter value is incorrect
- `TypeError` - sending a string when a function is expecting an int or calling a function with the wrong number or type of arguments
- `ImportError` - when Python can't find a module you're trying to import
- `FileNotFoundError` - when you try to perform file-related operations (opening, reading,

writing, or deleting) on a file or directory that does not exist

From <https://www.coursera.org/learn/automating-real-world-tasks-python/supplement/oXv8O/exception-handling>

Error budgets

Can you remember a time when you were growing up and someone gave you twenty dollars to spend at the grocery store? They might have said to bring home a gallon of milk and a dozen eggs, and whatever money is left you can spend however you want. That extra money — or the change — is akin to an error budget in service operations.
In this reading, you will learn more about error budgets, how they pertain to IT roles, and how an error budget can be used in the real world.

Definition of an error budget

An error budget is the maximum amount of time a software program can fail and still be in compliance with the service-level objective (SLO). An error budget is typically represented by a percentage. A simple example is this: If an SLO states that a website should function properly 99.9% of the time, then the error budget is only 0.1%.
Now let's calculate the error budget using time as a measurement in the question below. Here is an example in which an error budget is computed for a month's time frame using the following formula:
Error budget = Total time * (1-SLO)
What is the error budget, in minutes, with an SLO of 99.9% uptime over a month?
The total time is the total time in minutes for a given time period. (Assume a 30-day month for this example.) The SLO is the service-level objective represented as a decimal.
Total time = 30*24*60 = 43,200. This formula multiplies 30 days by 24 hours in each day by 60 minutes in each hour to get a total of 43,200 minutes.
SLO = 99.9/100 = 0.999. This value represents the SLO as a decimal. Substitute these values into the formula to get:
Error budget = 43,200 * (1–0.999)
Error budget = 43.1 minutes. This means the maximum amount of time the service can be down is up to 43 minutes per month without violating the agreed-upon reliability standards (the SLO).

The role of error budgets

Error budgets are part of cloud operations, site reliability engineering, and DevOps teams. They are used as a metric to make sure everything is running smoothly. If everything is running smoothly and there is a significant amount of time to use from the error budget, then DevOps members can use this time to invest in innovation on a product or software program. Error budgets also help establish limits for the development and programming teams. If there is not a lot of error budget to use, then developers know that they can't try new things and that their focus needs to remain on the reliability of the product or program. Developers should save the release of new features for when the error budget is large.

Key takeaways

Error budgets are typically represented as the maximum amount of time that a program is able to fail without violating an agreement. The error budget is based on the agreed-upon SLO between the clients and vendor. Developers are in favor of higher error budgets because this allows them to innovate and try new things within the product or service.
Mark as completed
Like
Dislike
Report an issue

From <https://www.coursera.org/learn/automating-real-world-tasks-python/supplement/EXXGj/error-budgets>

Congratulations! You've made it to the end of the module! Before you tackle the practice quiz and learn more about how DevOps is used in the real world, let's review everything you've learned about DevOps.
DevOps is a methodology that replaced the traditional way of developing and deploying code. DevOps is a collaborative effort between the development and operations teams. They work in an agile environment to eliminate the communication gap between the two teams. This allows the software to be developed more quickly and reliably. Automation is a key element through the entire DevOps process. DevOps has five key principles that include: collaboration, automation, continuous improvement, customer mindset, and end-goal-focused creation. For DevOps to be successful, team members need to adjust their mindsets and be open to working in a collaborative way with new processes.
A service-level agreement (SLA) acts as a promise and can be a legally binding agreement between a vendor and its clients or users. It includes the sale contract outlining the minimum level of performance for the client's or user's application, and it includes the consequences if there's a breach in the contract. Metrics in the SLA include uptime, responsiveness, and responsibilities.

Service-level objectives (SLOs) are found within an SLA and focus on specific metrics like uptime and response time. They should be measurable and clearly written. SLOs set the customers' expectations and communicate to the IT and DevOps teams the goals they need to achieve.
Service-level indicators (SLIs) measure the performance of an application against the SLOs at any given time. Monitoring SLIs provides you with valuable insights on whether your application is meeting the objective or violating the SLA. SLIs help development and operation teams analyze and determine if the promise — or promises — they made to their customers or users are upheld.
An error budget is the maximum amount of time a software program can fail and still be in compliance with the SLO. This value is typically represented in percentage form and is the difference between 100% and the SLO percentage value. For example, if an SLO states that a website should function properly 99.8% of the time, then the error budget is only 0.2%. DevOps teams use their error budget time either to focus on the reliability of the program or to take risks in the program (if it is running smoothly).

From <https://www.coursera.org/learn/automating-real-world-tasks-python/supplement/qXhX7/devops-review>

SLAs

The words *I promise* are powerful and meaningful. If you tell someone you promise to do something, that person is expecting you to follow through with your promise. In the operations and tech world, these promises are called service-level agreements (SLAs). In this reading, you will learn more about SLAs, how they relate to service-level objectives (SLOs), and common SLAs in the industry.

Service-level agreements

A service-level agreement is an agreement between a vendor and its clients or users that can be legally binding. Typically the SLA is included in the sale contract, guaranteeing a minimum level of performance for the client's or user's application. You can think of these as promises you are giving your users. And as with any promise there are consequences for breaking that promise. The SLA writes out the consequences that result if the promises are not met. Metrics in the SLA can include uptime, responsiveness, and responsibilities.

How SLAs relate to SLOs

An SLA outlines the specific SLOs and the penalties for violating them. Oftentimes, these consequences are a financial penalty or a cancellation of the contract. Because of this, it is important to understand what SLAs have been agreed upon to make informed decisions about how much to invest in reliability.

SLA examples

To view examples and templates of SLAs used in the industry, view the following:
Service-level agreement (SLA) examples and template

Key takeaways

An SLA is a service agreement typically between an IT service provider and a client. It outlines the details of the service, including what the service should accomplish and the consequences if there is a breach in the contract.

From <https://www.coursera.org/learn/automating-real-world-tasks-python/supplement/IrxrH/slas>

SLOs

All services have a level of quality and performance they are trying to obtain or reach. In the world of tech and operations, these are called service-level objectives (SLOs).
In this reading, you will learn more about SLOs, including how to write appropriate SLOs and how SLOs relate to service-level agreements (SLAs) and service-level indicators (SLIs).

Service-level objectives (SLOs), defined

SLOs are found within an SLA and focus on specific and measurable metrics like uptime and response time. SLOs set the customers' expectations and communicate to the IT and DevOps teams the goals they need to achieve. It's important to understand and educate yourself on the SLOs that have been promised in the SLA so you can make informed decisions about how to invest in reliability and keep your promise to achieve the objectives.
When creating SLOs, write them as simply and clearly as possible. If the SLOs are vague or immeasurable, they will not serve their purpose — and you will be setting yourself up for failure.

SLO example

Imagine that you work for an IT company that has entered into a contract with a customer, and that contract contains the following SLA:

You will maintain 99% uptime of your application, defined as "the home page loads correctly 99% of the time within 10 seconds or less."
The SLA tells you which SLIs need to be monitored, including the page load time and whether the page loads correctly or returns an HTTP error message.
Your SLOs include:

9. The page will load in 10 seconds or less 99% of the time.
10. The page will return an HTTP 200 (success) code 99% of the time.

Now that your SLA and SLOs are defined, you can use a combination of monitoring tools to test the site periodically and record the results for tracking your real-world performance against the SLOs.

Key takeaways

SLOs are important as they help achieve promises defined in the SLA. These help set customer expectations and should be clearly written, measurable, and attainable.
Mark as completed
Like
Dislike
Report an issue

From <https://www.coursera.org/learn/automating-real-world-tasks-python/supplement/fgEHK/slos>

SLIs

A common term used by DevOps engineers and software developers is service-level indicator (SLI). SLIs measure the performance of an application at any given time.
In this reading, you will learn more about SLIs, including what they are, how to measure and monitor them, and how they relate to service-level agreements (SLAs) and service-level objectives (SLOs).

Service-level indicators

An SLI measures the performance of your application against the objective, or SLO. It's important to monitor SLIs to determine if your application is meeting the objective — or violating the SLA. Simply, SLIs help answer the question, *"How did we do?"* in terms of having a promise and reaching that promise. Remember to keep your SLIs simple and choose the right metrics to track.

SLI example

Imagine that you work for an IT company and enter into a contract with customers. The contract contains the following SLA:
You will maintain 99% uptime of your application, defined as "the home page loads correctly 99% of the time within 10 seconds or less."
The SLA tells you which SLIs you should monitor, including the page load time and whether the page loads correctly or returns an HTTP error message.
Monitoring tools like DataDog or AppDynamics can measure and record these metrics for you. These tools offer the ability to perform synthetic checks, by simulating a user accessing your application, and recording the results. The results from the synthetics checks can be used as your SLIs. These monitoring tools help you determine if you are staying within your SLOs.

Key takeaways

Service-level indicators help make service-level objectives measurable. They provide actual data on performance. SLAs, SLOs, and SLIs all play a role in the overall service reliability.

From <https://www.coursera.org/learn/automating-real-world-tasks-python/supplement/XFhD3/slis>

Project Problem Statement

Okay, here's the scenario:
You work for an online fruit store, and you need to develop a system that will update the catalog information with data provided by your suppliers. When each supplier has new products for your store, they give you an image and a description of each product.
Given a bunch of images and descriptions of each of the new products, you'll:

- Upload the new products to your online store. Images and descriptions should be uploaded separately, using two different web endpoints.
- Send a report back to the supplier, letting them know what you imported.

Since this process is key to your business's success, you need to make sure that it keeps running! So, you'll also:

- Run a script on your web server to monitor system health.
- Send an email with an alert if the server is ever unhealthy.

Hopefully this summary has helped you start thinking about how you'll approach this task. In case you're feeling a little scared, don't worry, you can definitely do this! You have all the

necessary tools, and the lab description will go into a lot more detail of what you need to do.
Up next, we'll give you a few tips that can help you along the way.
Mark as completed
Like
Dislike
Report an issue

From <https://www.coursera.org/learn/automating-real-world-tasks-python/supplement/j28hX/project-problem-statement>