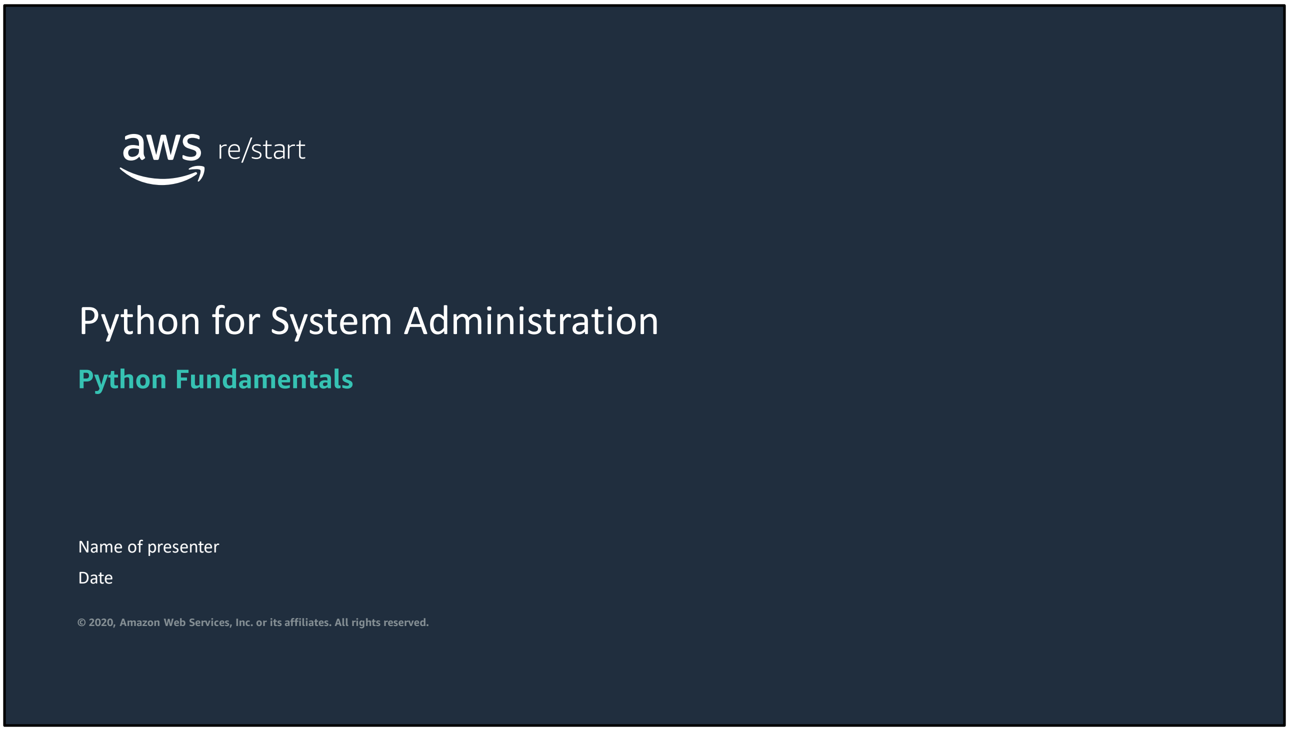


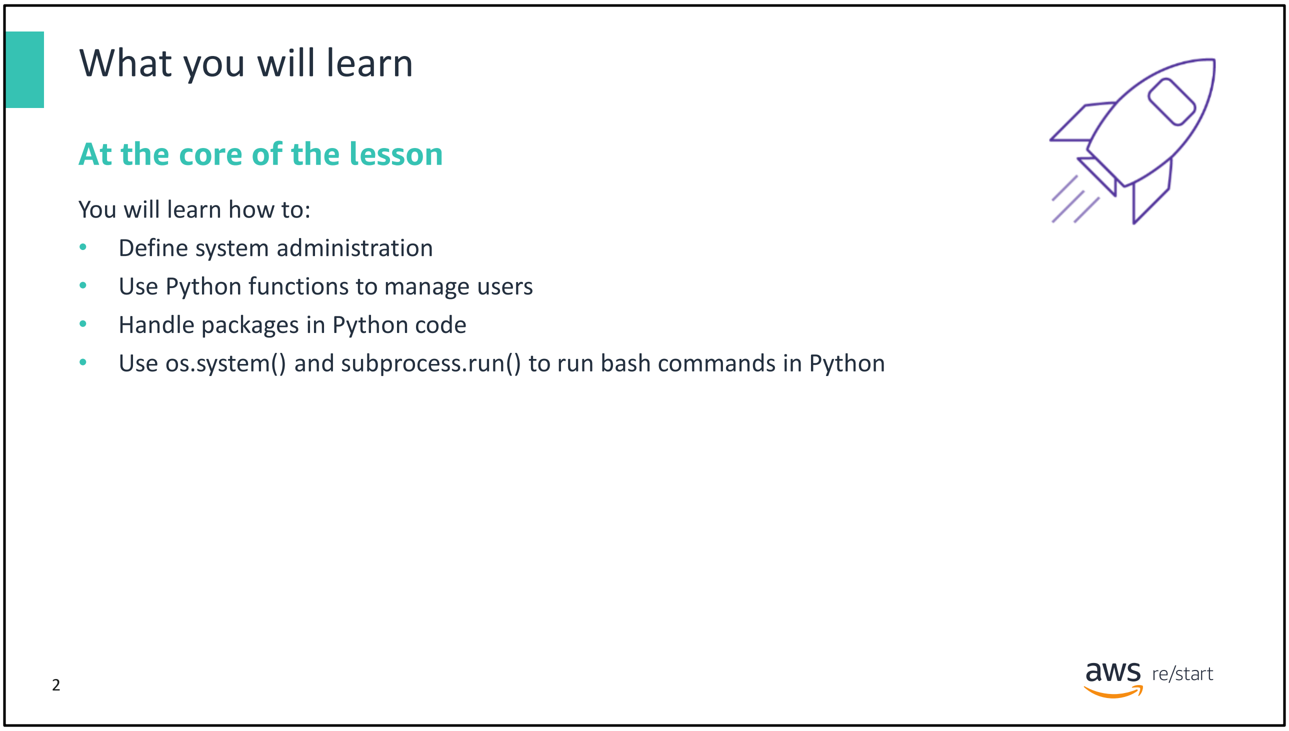
Python for System Administration

Python Fundamentals

Name of presenter

Date

Welcome to Python for System Administration.



What you will learn

At the core of the lesson

You will learn how to:

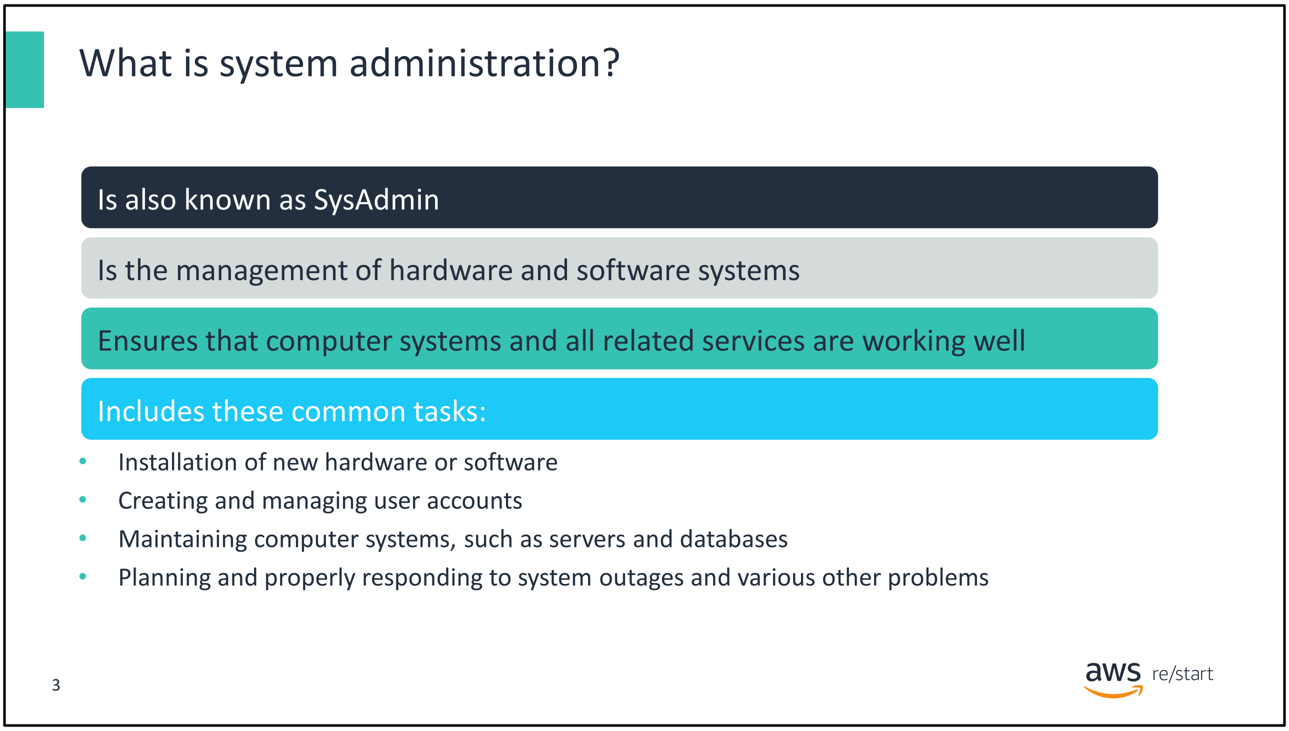
- Define system administration
- Use Python functions to manage users
- Handle packages in Python code
- Use os.system() and subprocess.run() to run bash commands in Python

2

aws re/start

In this module, you will learn how to:

- Define system administration
- Recognize how to manage users
- Recognize how to handle packages
- Recognize how to use os.system() and subprocess.run() to make complex decisions



What is system administration?

Is also known as SysAdmin

Is the management of hardware and software systems

Ensures that computer systems and all related services are working well

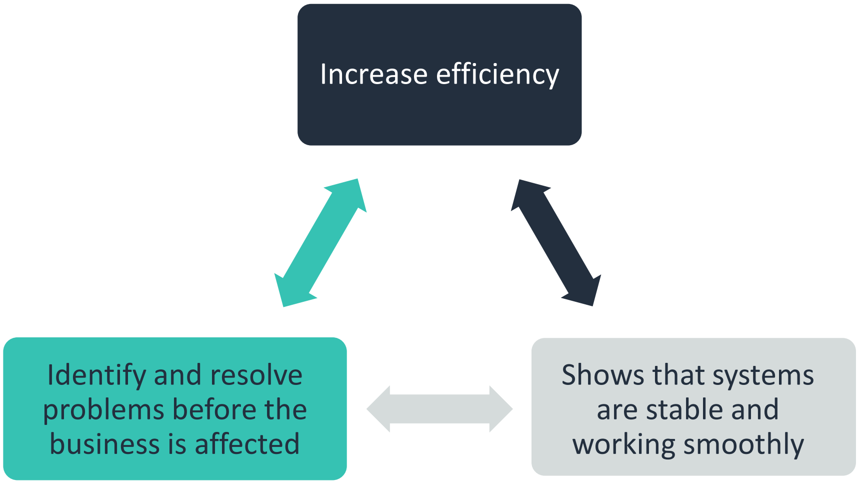
Includes these common tasks:

- Installation of new hardware or software
- Creating and managing user accounts
- Maintaining computer systems, such as servers and databases
- Planning and properly responding to system outages and various other problems

Discussion question

- How could you use Python to make system administration tasks easier?

Benefits of SysAdmin



Increase efficiency

Identify and resolve problems before the business is affected

Shows that systems are stable and working smoothly

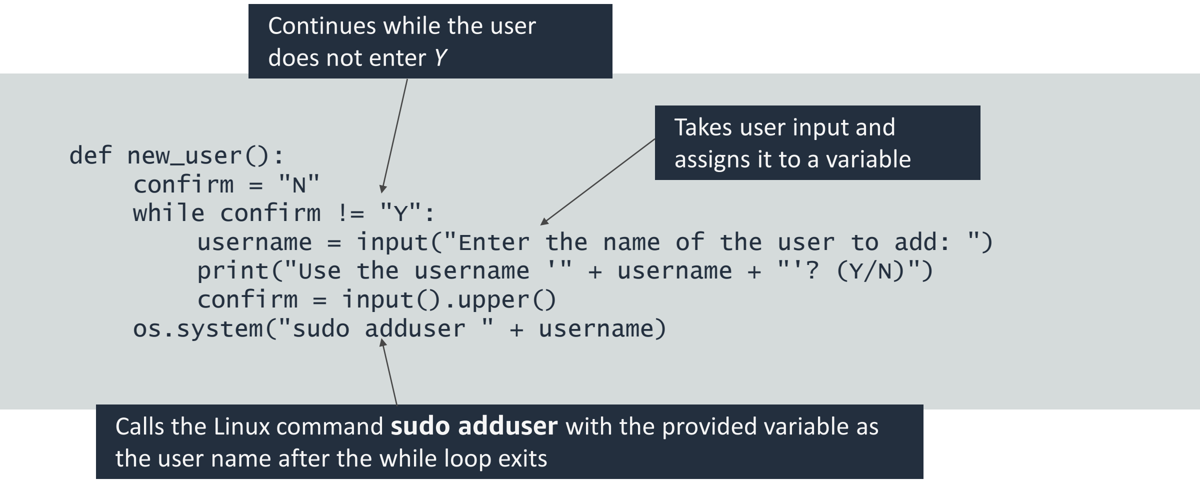
Activity: Working with Users

The following slides contain code snippets for managing users. Use your current knowledge of Python programming to read the code and decipher what it does.

Activity: Adding a user

```
def new_user():
    confirm = "N"
    while confirm != "Y":
        username = input("Enter the name of the user to add: ")
        print("Use the username '" + username + "'? (Y/N)")
        confirm = input().upper()
    os.system("sudo adduser " + username)
```

Activity: Adding a user – solution



Continues while the user does not enter *Y*

Takes user input and assigns it to a variable

```
def new_user():
    confirm = "N"
    while confirm != "Y":
        username = input("Enter the name of the user to add: ")
        print("Use the username '" + username + "'? (Y/N)")
        confirm = input().upper()
    os.system("sudo adduser " + username)
```

Calls the Linux command **sudo adduser** with the provided variable as the user name after the while loop exits

Activity: Removing a user

```
def remove_user():
    confirm = "N"
    while confirm != "Y":
        username = input("Enter the name of the user to remove: ")
        print("Remove the user : '" + username + "'? (Y/N)")
        confirm = input().upper()
    os.system("sudo userdel -r " + username)
```

Activity: Removing a user – solution

Continues while the user does not enter *Y*

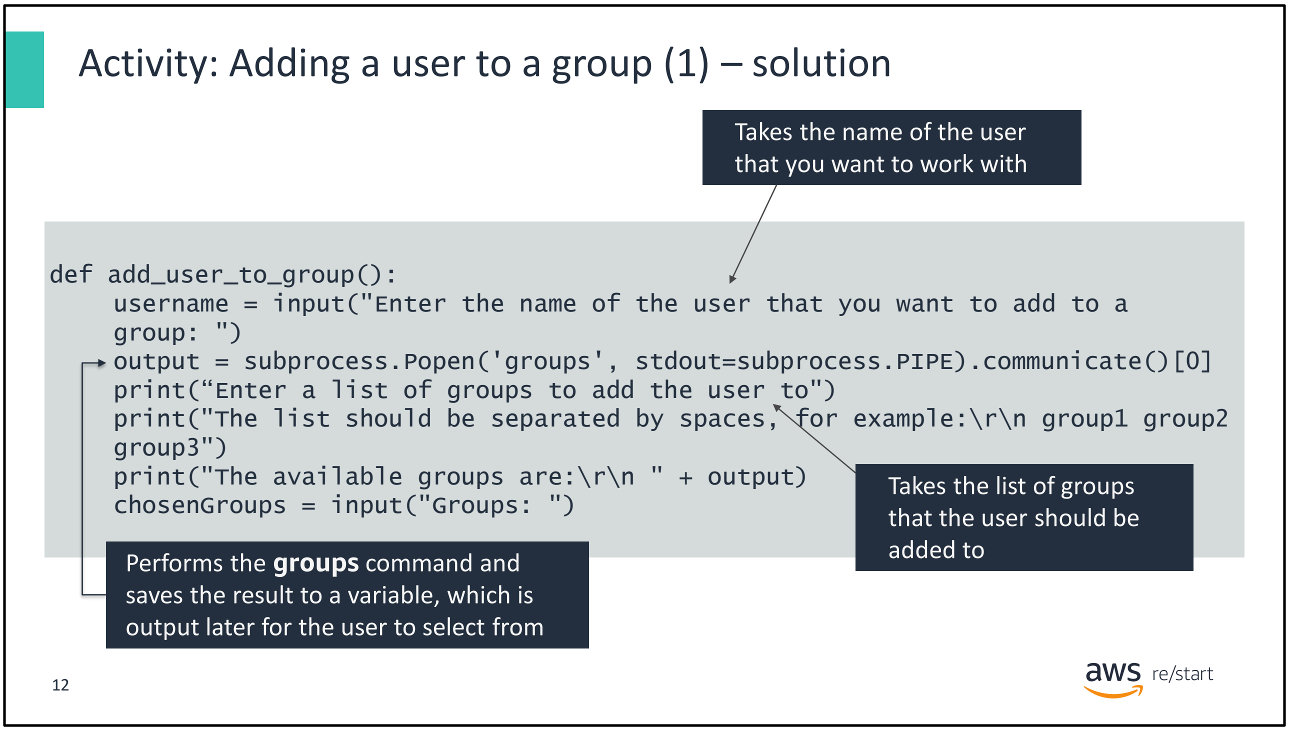
Takes user input and assigns it to a variable

```
def remove_user():
    confirm = "N"
    while confirm != "Y":
        username = input("Enter the name of the user to remove: ")
        print("Remove the user : '" + username + "'? (Y/N)")
        confirm = input().upper()
    os.system("sudo userdel -r " + username)
```

Calls the Linux command **sudo userdel -r** with the provided variable as the user name after the while loop exits

Activity: Adding a user to a group (1)

```
def add_user_to_group():
    username = input("Enter the name of the user that you want to add to a
    group: ")
    output = subprocess.Popen('groups', stdout=subprocess.PIPE).communicate()[0]
    print("Enter a list of groups to add the user to")
    print("The list should be separated by spaces, for example:\r\n group1 group2
    group3")
    print("The available groups are:\r\n " + output)
    chosenGroups = input("Groups: ")
```



Activity: Adding a user to a group (1) – solution

Takes the name of the user that you want to work with

```
def add_user_to_group():
    username = input("Enter the name of the user that you want to add to a
    group: ")
    output = subprocess.Popen('groups', stdout=subprocess.PIPE).communicate()[0]
    print(“Enter a list of groups to add the user to")
    print("The list should be separated by spaces, for example:\r\n group1 group2
    group3")
    print("The available groups are:\r\n " + output)
    chosenGroups = input("Groups: ")
```

Takes the list of groups that the user should be added to

Performs the **groups** command and saves the result to a variable, which is output later for the user to select from

aws re/start

Activity: Adding a user to a group (2)

```
output = output.split(" ")
chosenGroups = chosenGroups.split(" ")
print("Add To:")
found = True
groupString = ""
```

Activity: Adding a user to a group (2) – solution

Splits the string from the previous section into an array

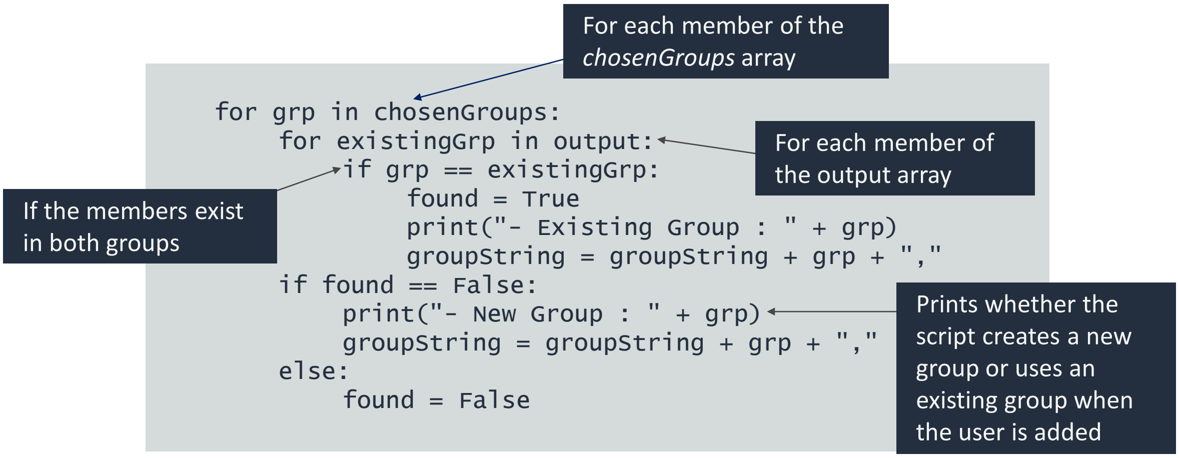
Splits the string from the previous section into an array

```
output = output.split(" ")
chosenGroups = chosenGroups.split(" ")
print("Add To:")
found = True
groupString = ""
```

Activity: Adding a user to a group (3)

```
for grp in chosenGroups:
    for existingGrp in output:
        if grp == existingGrp:
            found = True
            print("- Existing Group : " + grp)
            groupString = groupString + grp + ","
    if found == False:
        print("- New Group : " + grp)
        groupString = groupString + grp + ","
    else:
        found = False
```

Activity: Adding a user to a group (3) – solution



```
for grp in chosenGroups:
    for existingGrp in output:
        if grp == existingGrp:
            found = True
            print("- Existing Group : " + grp)
            groupString = groupString + grp + ","
    if found == False:
        print("- New Group : " + grp)
        groupString = groupString + grp + ","
    else:
        found = False
```

For each member of the *chosenGroups* array

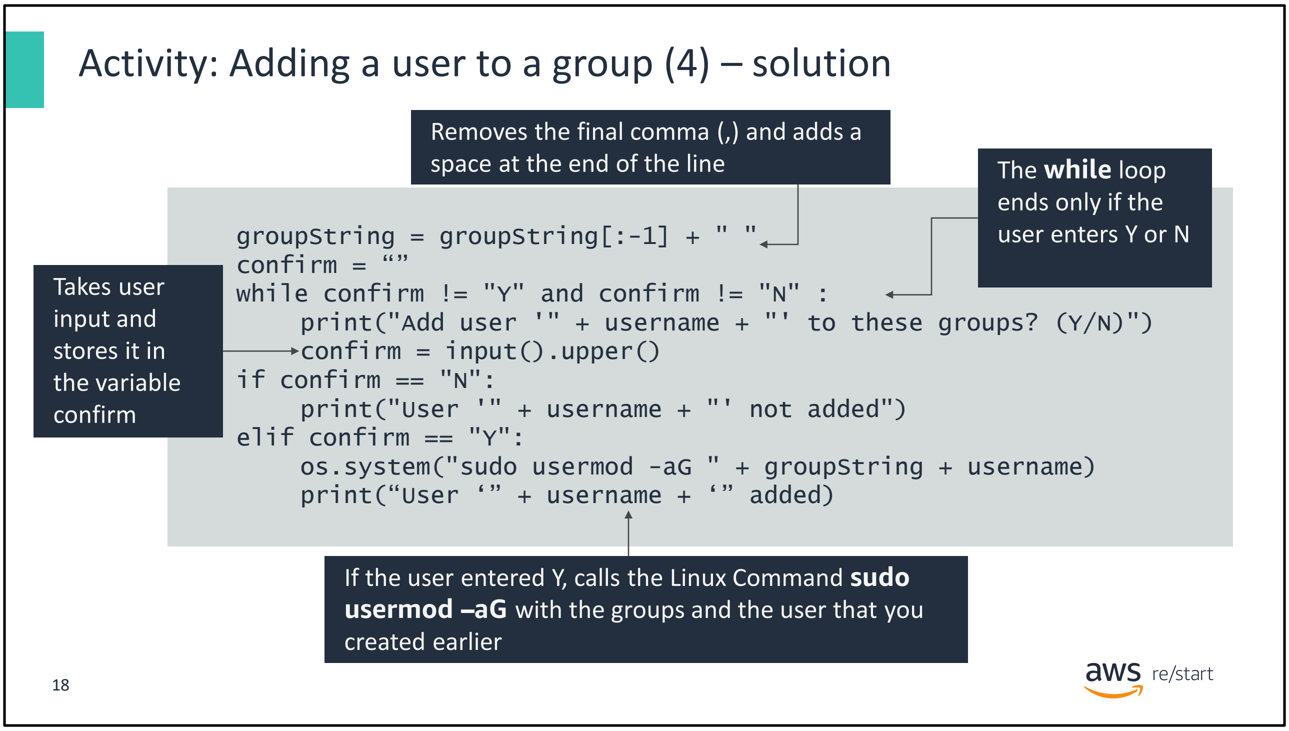
For each member of the output array

If the members exist in both groups

Prints whether the script creates a new group or uses an existing group when the user is added

Activity: Adding a user to a group (4)

```
groupString = groupString[:-1] + " "
confirm = “”
while confirm != "Y" and confirm != "N" :
     print("Add user '" + username + "' to these groups? (Y/N)")
     confirm = input().upper()
if confirm == "N":
     print("User '" + username + "' not added")
elif confirm == "Y":
     os.system("sudo usermod -aG " + groupString + username)
     print(“User ‘” + username + ‘” added)
```



Activity: Adding a user to a group (4) – solution

Removes the final comma (,) and adds a space at the end of the line

The **while** loop ends only if the user enters Y or N

Takes user input and stores it in the variable confirm

```
groupString = groupString[:-1] + " "
confirm = “”
while confirm != "Y" and confirm != "N" :
    print("Add user '" + username + "' to these groups? (Y/N)")
    confirm = input().upper()
if confirm == "N":
    print("User '" + username + "' not added")
elif confirm == "Y":
    os.system("sudo usermod -aG " + groupString + username)
    print(“User ‘” + username + ‘” added)
```

If the user entered Y, calls the Linux Command **sudo usermod –aG** with the groups and the user that you created earlier

The **while** loop exits if the user enters **n**, **N**, **y**, or **Y**. The **input.upper()** converts the user input to upper case.

Activity: Handling Packages

The following slides contain code snippets for package management. Use your current knowledge of Python programming to read the code and decipher what it is does.

Activity: Handling packages (1)

```
def install_or_remove_packages():
    iOrR = “”
    while iOrR != “I” and iOrR != “R”:
        print("Would you like to install or remove packages? (I/R)")
        iOrR = input().upper()
    if iOrR == "I":
        iOrR = "install"
    elif iOrR == "R":
        iOrR = "remove"
```

Activity: Handling packages (1) – solution

```
def install_or_remove_packages():
    iOrR = “”
    while iOrR != “I” and iOrR != “R”:
        print("Would you like to install or remove packages? (I/R)")
        iOrR = input().upper()
    if iOrR == "I":
        iOrR = "install"
    elif iOrR == "R":
        iOrR = "remove"
```

Checks whether the user wants to install or remove packages

Activity: Handling packages (2)

```
print("Enter a list of packages to install")
print("The list should be separated by spaces, for example:")
print(" package1 package2 package3")
print("Otherwise, input 'default' to " + iOrR + " the default packages listed in this program")
packages = input().lower()
if packages == "default":
      packages = defaultPackages
if iOrR == "install":
      os.system("sudo apt-get install " + packages)
```

Activity: Handling packages (2) – solution

Describes how the input should be formatted

```
print("Enter a list of packages to install")
print("The list should be separated by spaces, for example:")
print(" package1 package2 package3")
print("Otherwise, input 'default' to " + iOrR + " the default packages listed in this program")
packages = input().lower()
if packages == "default":
      packages = defaultPackages
if iOrR == "install":
      os.system("sudo apt-get install " + packages)
```

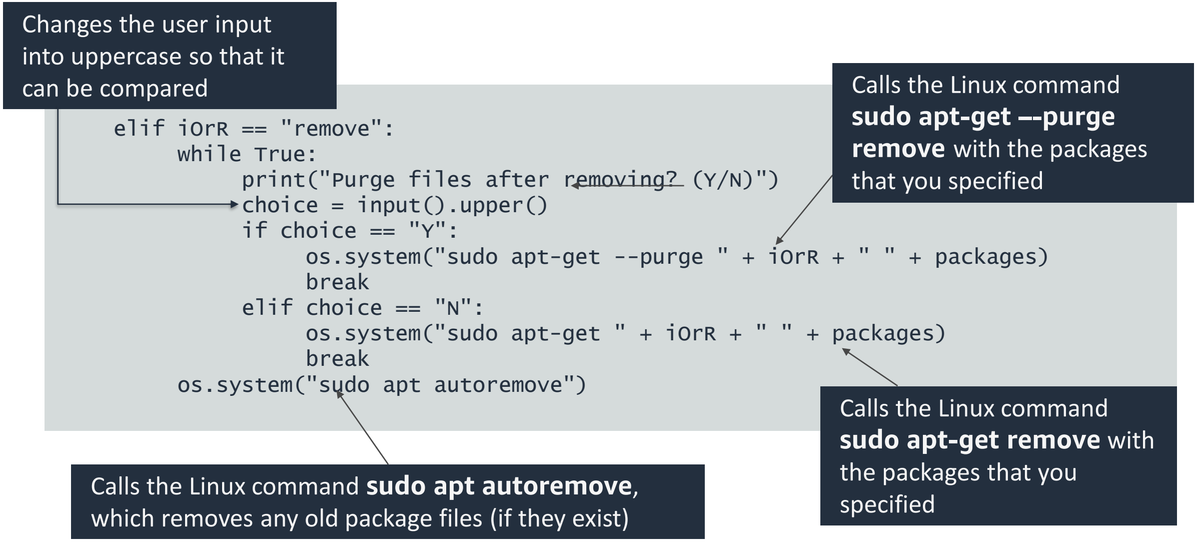
Calls the Linux command **sudo apt-get install** with the packages that you specified

Installs the default list of packages for the script if the user specifies ***default***

Activity: Handling packages (3)

```
elif iOrR == "remove":
     while True:
          print("Purge files after removing? (Y/N)")
          choice = input().upper()
          if choice == "Y":
               os.system("sudo apt-get --purge " + iOrR + " " + packages)
               break
          elif choice == "N":
               os.system("sudo apt-get " + iOrR + " " + packages)
               break
     os.system("sudo apt autoremove")
```

Activity: Handling packages (3) – solution



Changes the user input into uppercase so that it can be compared

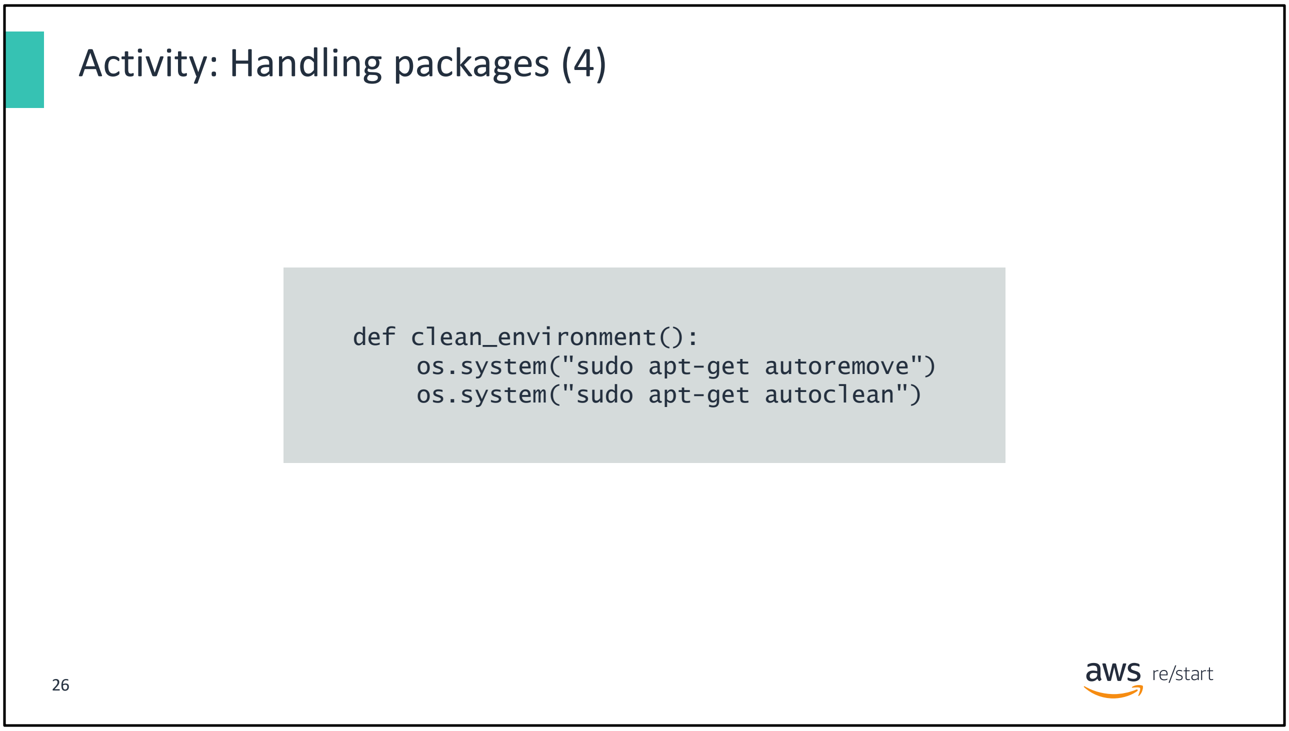
```
elif iOrR == "remove":
     while True:
          print("Purge files after removing? (Y/N)")
          choice = input().upper()
          if choice == "Y":
               os.system("sudo apt-get --purge " + iOrR + " " + packages)
               break
          elif choice == "N":
               os.system("sudo apt-get " + iOrR + " " + packages)
               break
     os.system("sudo apt autoremove")
```

Calls the Linux command **sudo apt-get –-purge remove** with the packages that you specified

Calls the Linux command **sudo apt-get remove** with the packages that you specified

Calls the Linux command **sudo apt autoremove**, which removes any old package files (if they exist)

aws re/start



Activity: Handling packages (4)

```
def clean_environment():
    os.system("sudo apt-get autoremove")
    os.system("sudo apt-get autoclean")
```

Activity: Handling packages (4) – solution

Removes dependencies that were installed with applications and are no longer used by anything on the system

```
def clean_environment():
    os.system("sudo apt-get autoremove")
    os.system("sudo apt-get autoclean")
```

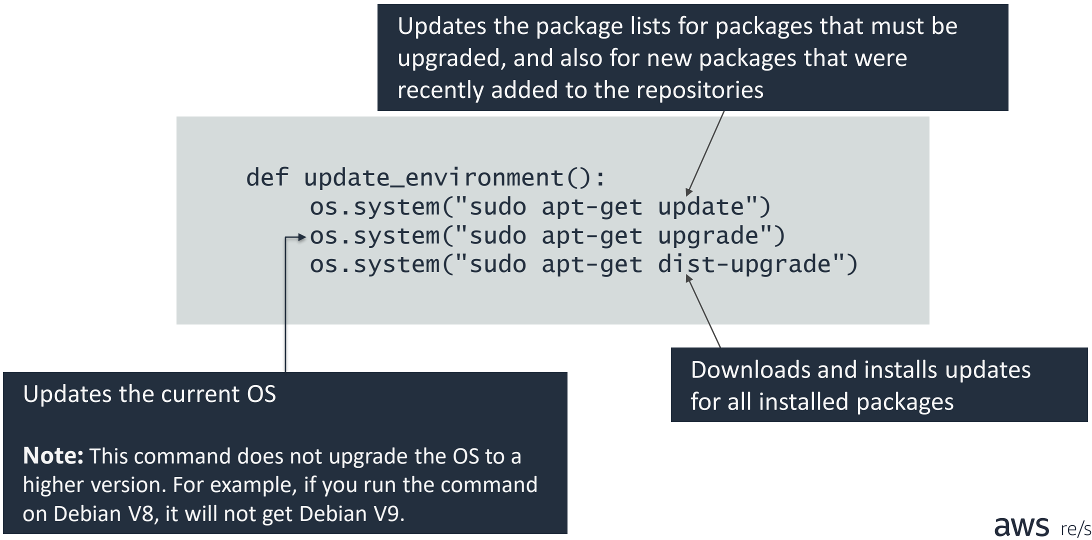
Cleans obsolete deb-packages

Used together, these two Linux commands are a good way to maintain an up-to-date and clean environment.

Activity: Handling packages (5)

```
def update_environment():
    os.system("sudo apt-get update")
    os.system("sudo apt-get upgrade")
    os.system("sudo apt-get dist-upgrade")
```

Activity: Handling packages (5) – solution



Updates the package lists for packages that must be upgraded, and also for new packages that were recently added to the repositories

```
def update_environment():
    os.system("sudo apt-get update")
    os.system("sudo apt-get upgrade")
    os.system("sudo apt-get dist-upgrade")
```

Updates the current OS

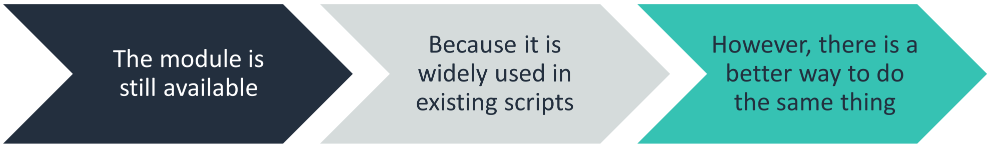
Note: This command does not upgrade the OS to a higher version. For example, if you run the command on Debian V8, it will not get Debian V9.

Downloads and installs updates for all installed packages

A better os.system(): subprocess.run()

In Python V3, the **os** module has been deprecated and replaced by the **subprocess** module.

Module deprecation:

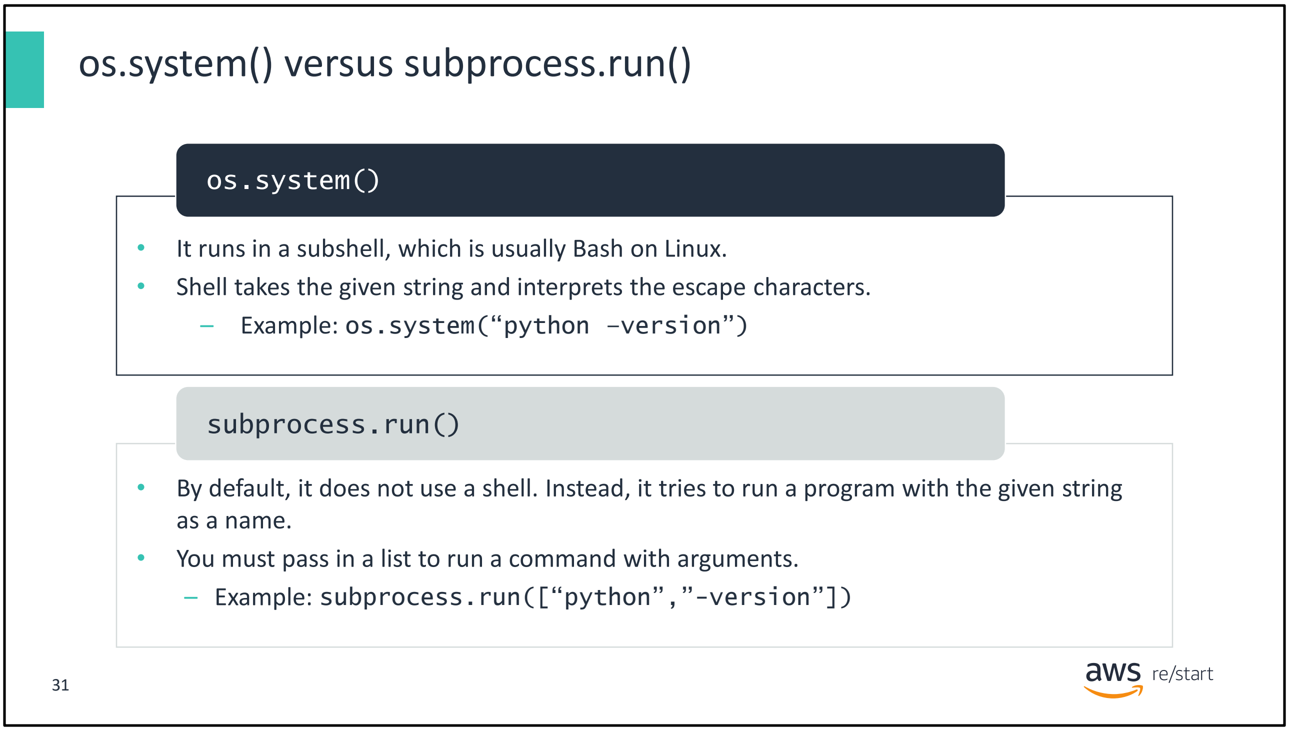


The module is still available

Because it is widely used in existing scripts

However, there is a better way to do the same thing

The equivalent function to `os.system()` is `subprocess.run()`.



os.system() versus subprocess.run()

`os.system()`

- It runs in a subshell, which is usually Bash on Linux.
- Shell takes the given string and interprets the escape characters.
  - Example: `os.system(“python –version”)`

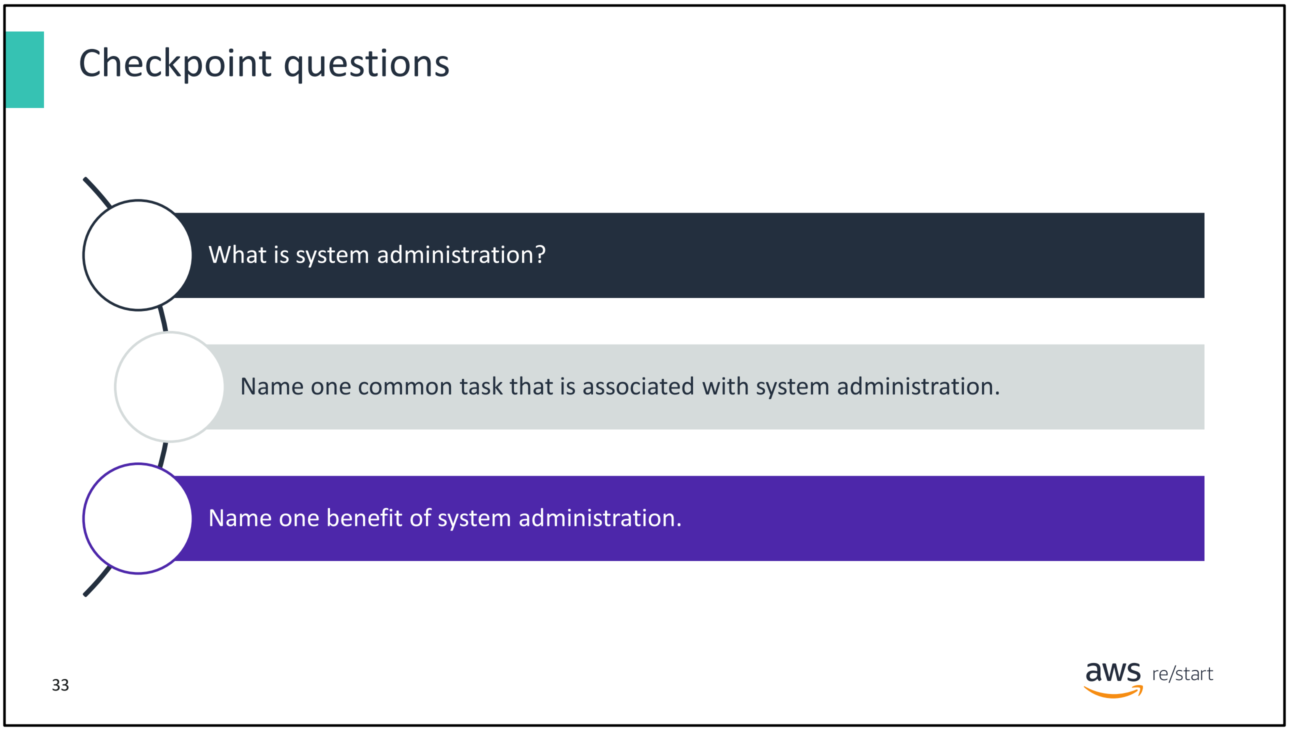
`subprocess.run()`

- By default, it does not use a shell. Instead, it tries to run a program with the given string as a name.
- You must pass in a list to run a command with arguments.
  - Example: `subprocess.run([“python”,”-version”])`

Why is subprocess.run() better than os.system()?

`subprocess.run()` is better than `os.system()` for the following reasons:

Safety	Developers often pass an input string to `os.system()` without checking the actual commands. This practice can be dangerous. For example, a malicious user can pass in a string to delete your files.
Separate process	`subprocess.run()` is implemented by a class that is called *Popen*, which is run as a separate process.
Additional functionality	Because `subprocess.run()` is really the Popen class, it has useful, new methods such as `poll()`, `wait()`, and `terminate()`.



Checkpoint questions

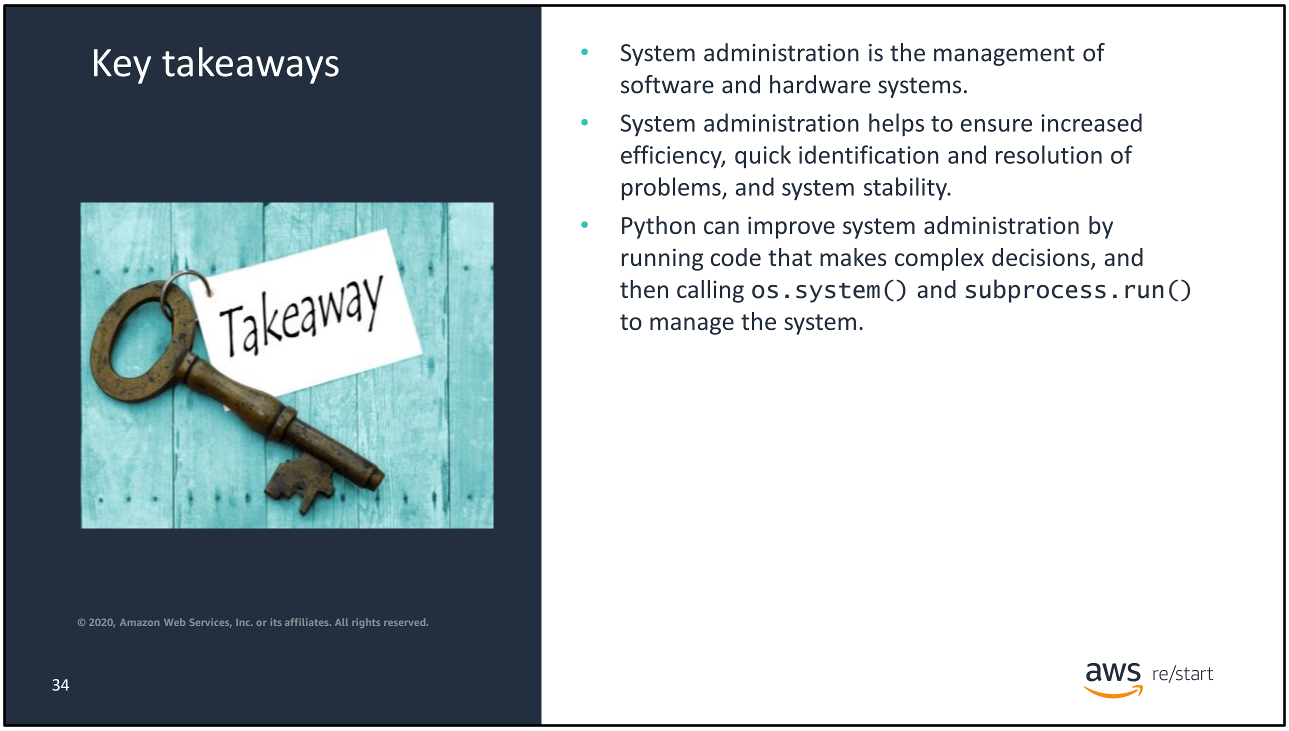
What is system administration?

Name one common task that is associated with system administration.

Name one benefit of system administration.

Answers:

1. System administration is the management of hardware or software systems, including configuration, upgrades, reliability, and security.
2. Some tasks that are associated with system administration include –
   - Performing backups
   - Archiving log files
   - Automating repetitive tasks
3. System administration keeps systems running smoothly, and helps to ensure that IT resources are available to meet business needs.



Key takeaways

- System administration is the management of software and hardware systems.
- System administration helps to ensure increased efficiency, quick identification and resolution of problems, and system stability.
- Python can improve system administration by running code that makes complex decisions, and then calling `os.system()` and `subprocess.run()` to manage the system.

aws re/start

Some key takeaways from this lesson include:

- System administration is the management of software and hardware systems.
- System administration helps to ensure increased efficiency, quick identification and resolution of problems, and system stability.
- Python can improve system administration by running code that makes complex decisions, and then calling `os.system()` and `subprocess.run()` to manage the system.