Welcome to Introduction to Programming.

What is programming?

What you will learn

At the core of the lesson

You will learn how to:

- Define programming
- Explain the function and features of an integrated development environment (IDE)
- Describe the cycle of programming

aws re/start

In this lesson, you will learn how to:

- Define programming
- Explain the function and features of an integrated development environment (IDE)
- Describe the cycle of programming

A purpose for programming: Automation

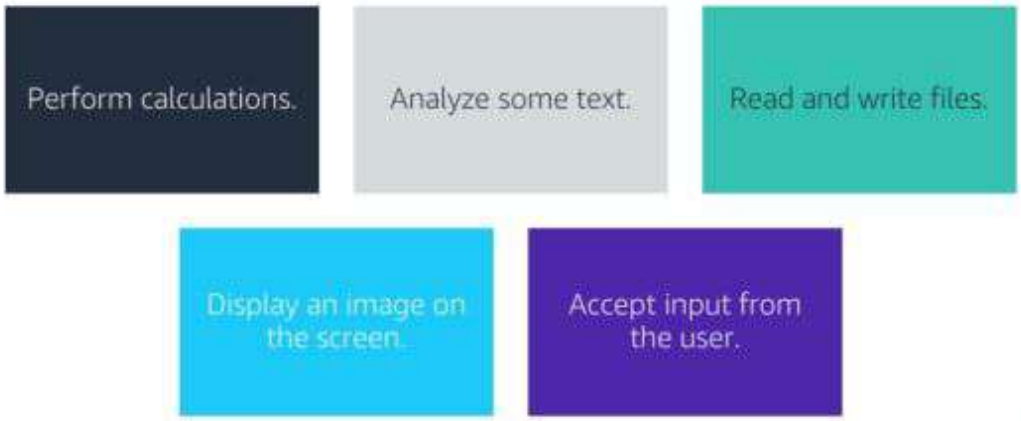
Automation refers to any technology that removes human interaction from a system, equipment, or process.

Scripts are often written to automate labor-intensive tasks and streamline workflows.

Before you automate a process, consider the following:

What is a programming language?

A programing language is a language that communicates instructions to a computer.
Possible instructions are:



How is software written?

Software is written by using text files.

Software is written by using a computer language.

Software is written by using text files

- Some text editors* include features that help programmers write code.
- Examples:
  - Microsoft Visual Studio Code
  - Sublime Text
  - Vi or Vim
  - nano
  - GNU emacs
  - Notepad++
  - TextEdit

Software is written by using text files.

Software is written by using a computer language.

***Microsoft Word is not a text editor.**

Software is written by using a computer language

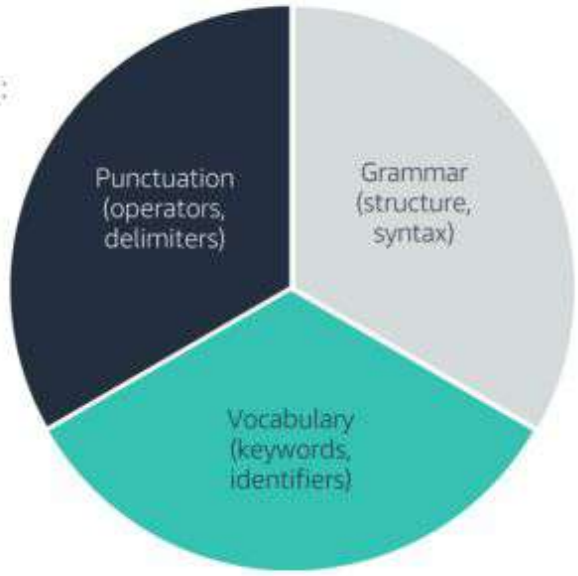
- Many different languages exist
- Each language has its own:
  - Grammar and syntax
  - Common uses
  - Community
- Examples:
  - Python
  - JavaScript
  - C#
  - C/C++

Software is written by using text files.

Software is written by using a computer language.

Programming language elements

Like human languages, programming languages have:



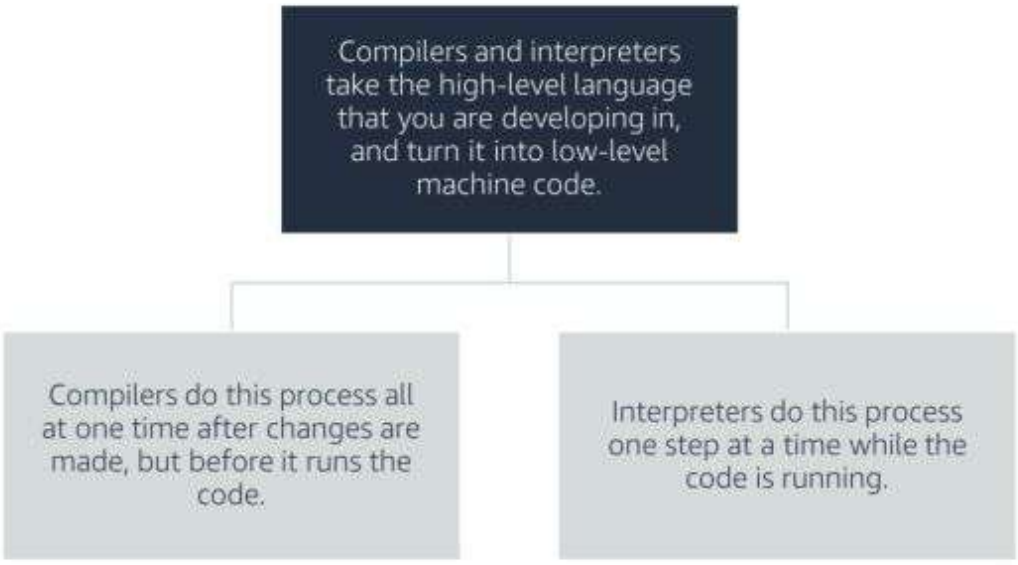
Integrated development environment (IDE)

An IDE can tell you which words are spelled incorrectly, which phrases are unclear, and syntax that you wrote incorrectly.

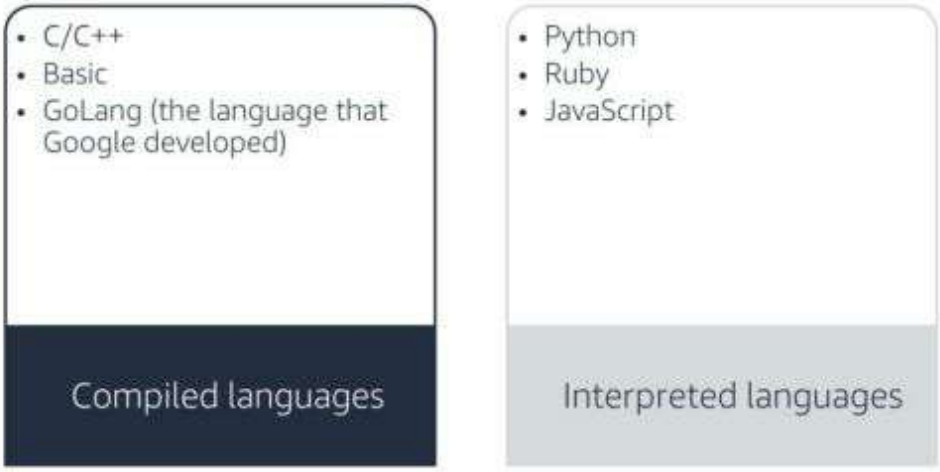
Most IDEs will suggest a fix for the issue.

Some IDEs work with only one language, such as PyCharm for Python. It will not show syntax errors for Java or C# to you.

Compilers and interpreters



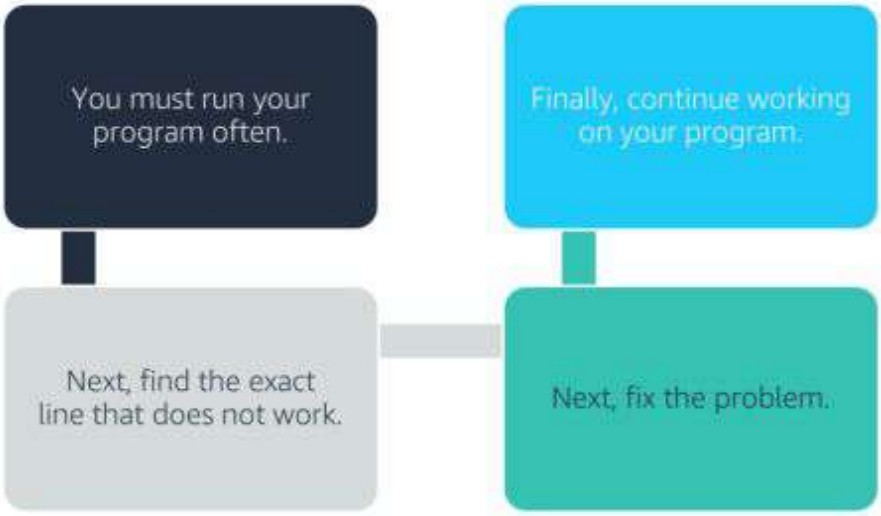
Compiled and interpreted languages



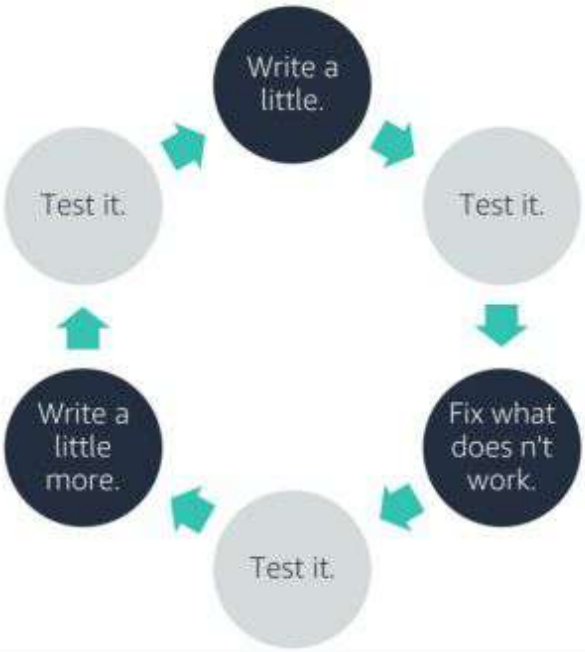
aws re/start

JavaScript is *interpreted* when it is loaded into a webpage.
Java is *compiled* before the program runs on a computer.

Run, debug, and correct software



Software is written iteratively



Categorize a value as a data type

What you will learn

At the core of the lesson

You will learn how to:

- Define the term *data type*
- Correctly match data types to data
- Assign values to variables and variables to data types

aws re/start

In this lesson, you will learn how to:

- Define the term *data type*
- Correctly match data types to data
- Assign values to variables and variables to data types

What is a data type?

A data type is the classification of a value that tells the computer how the programmer intends the data to be interpreted.

Examples:

Data Value	Data Type
45	Integer
290578L	Long
1.02	Float
True	Boolean
"My dog is on the bed."	String
"45"	String

What is a data type?

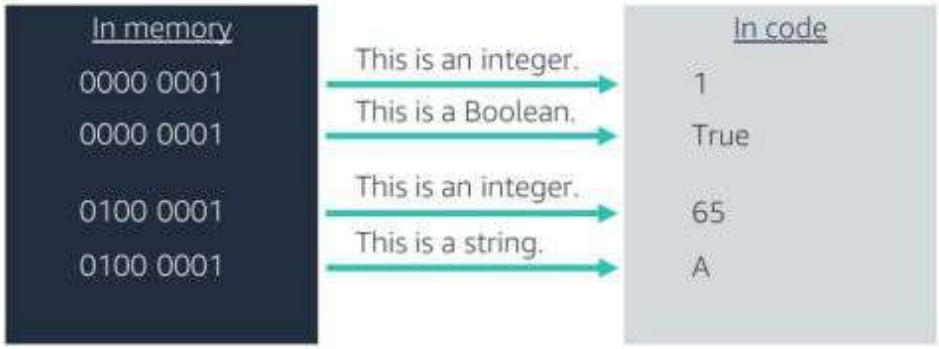
A data type is the classification of a value that tells the computer how the programmer intends the data to be interpreted.

Examples:

Data Value	Data Type
45	Integer
290578L	Long
1.02	Float
True	Boolean
"My dog is on the bed."	String
"45"	String

Why must the type of data be tagged?

- In memory, everything consists of 0s and 1s.
- Data typing tells the computer how to:
  - Encode a data value into memory
  - Decode a data value out of memory



Activity: Identify Data Types

19

Instructions

1. For each value, identify the data type:

    1. "The Martian"
    2. 1.618
    3. 10082L
    4. False
    5. "True"

aws re/start

"The Martian" - String
1.618 - Float
10082L - Long
False - Boolean
"True" - String

What is a variable?

- A variable is an identifier in your code that represents a value in memory.
- The variable name helps humans to remember what the value means.

Assigning a variable to a value

- Nearly all languages have an *assignment operator*.
- Most languages use the equal sign (=).

Python
isCoder = True

VB.NET
daysOnJob = 1

F#
daysOnJob <- 1

Pascal
isCoder := true

Assigning a data type to a variable

- Some languages **expect** the data type to be included when first using the variable.

Objective C
int daysOnJob = 1

Java
boolean isCoder = true

Assigning a data type to a variable, continued

- Some languages *infer* the data type based on the value that is assigned.

Python
count = 10

Swift
var daysOnJob = 1

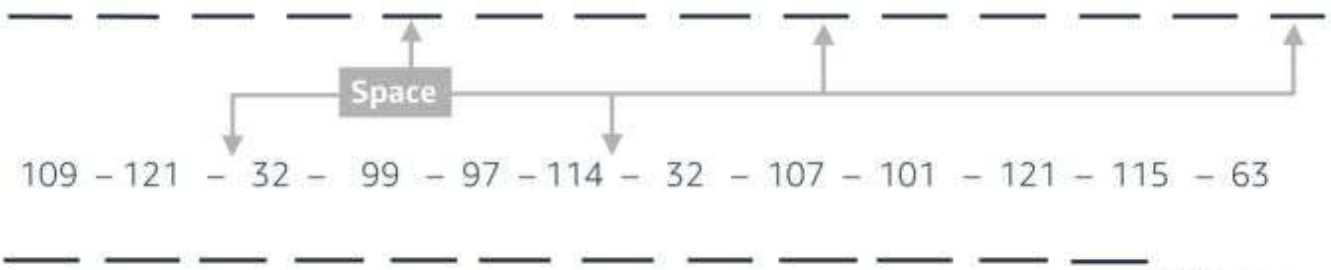
TypeScript
let isCoder = true

Activity: Represent letters as numbers

American Standards Association for Information Interchange (ASCII) is a system that associates encoding characters into computers.

Use the ASCII table at **ASCII Table** to decipher the following message:

72 – 97 – 118 – 101 – 32 – 121 – 111 – 117 – 32 – 115 – 101 – 101 – 110 – 32



109 – 121 – 32 – 99 – 97 – 114 – 32 – 107 – 101 – 121 – 115 – 63

Activity: Represent letters as numbers

American Standards Association for Information Interchange (ASCII) is a system that associates encoding characters into computers.

Use the ASCII table at **ASCII Table** to decipher the following message:

72 – 97 – 118 – 101 – 32 – 121 – 111 – 117 – 32 – 115 – 101 – 101 – 110 – 32

H a v e _ y o u _ s e e n _

109 – 121 – 32 – 99 – 97 – 114 – 32 – 107 – 101 – 121 – 115 – 63

m y _ c a r _ k e y s ?

aws re/start

Answer: Have you seen my car keys?

Combine values into composite data types

What you will learn

At the core of the lesson

You will learn how to:

- Define composite data types
- Identify composite data types from a list of properties
- List the attributes of a function
- Define different types of collections
- Combine values into a composite data type

aws re/start

In this lesson, you will learn how to:

- Define composite data types
- Identify composite data types from a list of properties
- List the attributes of a function
- Define different types of collections
- Combine values into a composite data type

What is a composite data type?

Until now, you have used primitive data types.

Primitive data type	• Data types that are built into a coding language with no modification
Composite data type	• Combines multiple data types into a single unit

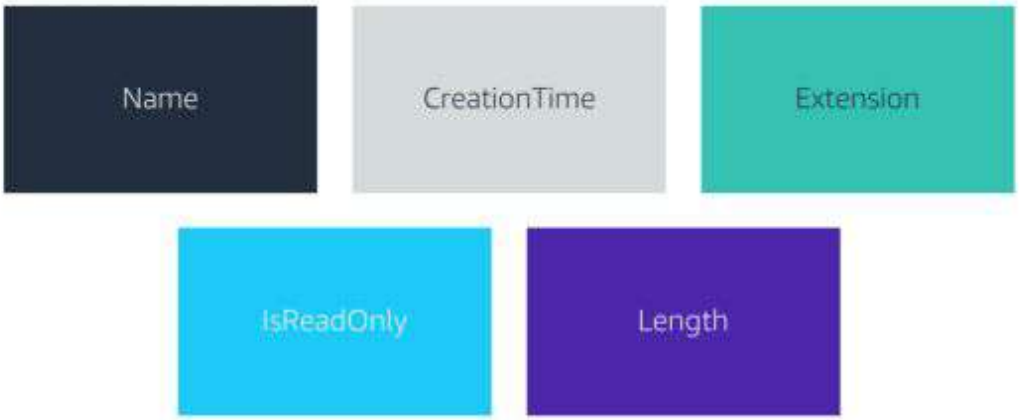
All modern programming languages have a way to create composite data types.

Composite data type example: Movie

Each movie could be described with:

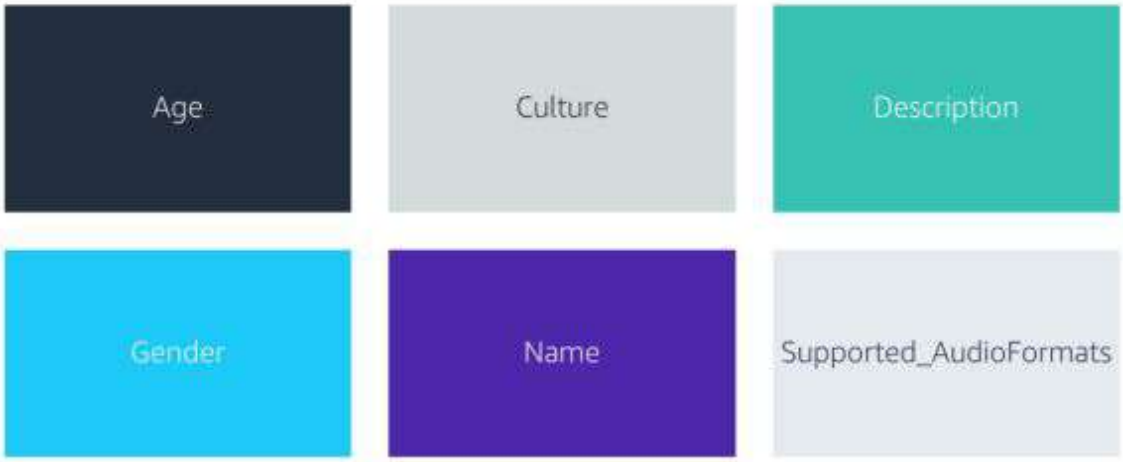
- A *name* (data type would be a *string*)
- A *year* that it was released (data type would be a *number*, or *integer*)
- A *flag* to indicate whether you have seen it or not (data type would be a *Boolean*)

Composite data type 1



Composite data type 2

Composite data type 3



Functions

Functions

Functions do something useful.

Functions can return a value (to be stored in a variable).

Functions can return a value based on input values.

Functions can accept values as input.

Functions can accept many values as input.

Or, a developer can create a composite data type.

Composite data types can be returned.

Composite data types can be in an array.

- The following slides provide examples of each of these attributes.

Functions, continued

- Functions do something useful.
- Functions can return a value (to be stored in a variable).
- Functions can return a value based on input values.
- Functions can accept values as input.
- Functions can accept many values as input.
- Or, a developer can create a composite data type.
- Composite data types can be returned.
- Composite data types can be in an array.

```
clearScreen()
```

Functions, continued

Functions do something useful.

Functions can return a value (to be stored in a variable).

```
pi = calculatePi()
```

Functions can return a value based on input values.

Functions can accept values as input.

Functions can accept many values as input.

Or, a developer can create a composite data type.

Composite data types can be returned.

Composite data types can be in an array.

Functions, continued

Functions do something useful.

Functions can return a value (to be stored in a variable).

Functions can return a value based on input values.

```
area = calculateArea(pi, 4)
```

Functions can accept values as input.

Functions can accept many values as input.

Or, a developer can create a composite data type.

Composite data types can be returned.

Composite data types can be in an array.

Functions, continued

Functions do something useful.

Functions can return a value (to be stored in a variable).

Functions can return a value based on input values.

Functions can accept values as input.

```
showValue(area)
```

Functions can accept many values as input.

Or, a developer can create a composite data type.

Composite data types can be returned.

Composite data types can be in an array.

Functions, continued

Functions do something useful.

Functions can return a value (to be stored in a variable).

Functions can return a value based on input values.

Functions can accept values as input.

Functions can accept many values as input.

```
fly(lat, lon, spd, hdg,
vspd, wspd, wdir)
```

Or, a developer can create a composite data type.

Composite data types can be returned.

Composite data types can be in an array.

Functions, continued

Functions do something useful.

Functions can return a value (to be stored in a variable).

Functions can return a value based on input values.

Functions can accept values as input.

Functions can accept many values as input.

Or, a developer can create a composite data type.

Composite data types can be returned.

Composite data types can be in an array.

```
fly(aircraft)

latitude
longitude
speed
heading
verticalSpeed
windSpeed
windDirection
```

Functions, continued

Functions do something useful.

Functions can return a value (to be stored in a variable).

Functions can return a value based on input values.

Functions can accept values as input.

Functions can accept many values as input.

Or, a developer can create a composite data type.

Composite data types can be returned.

Composite data types can be in an array.

```
aircraftFuture =
predict(aircraftNow, 60)
```

```
latitude
longitude
speed
heading
verticalSpeed
windSpeed
windDirection
```

aws re/start

Functions, continued

Functions do something useful.

Functions can return a value (to be stored in a variable).

Functions can return a value based on input values.

Functions can accept values as input.

Functions can accept many values as input.

Or, a developer can create a composite data type.

Composite data types can be returned.

Composite data types can be in an array.

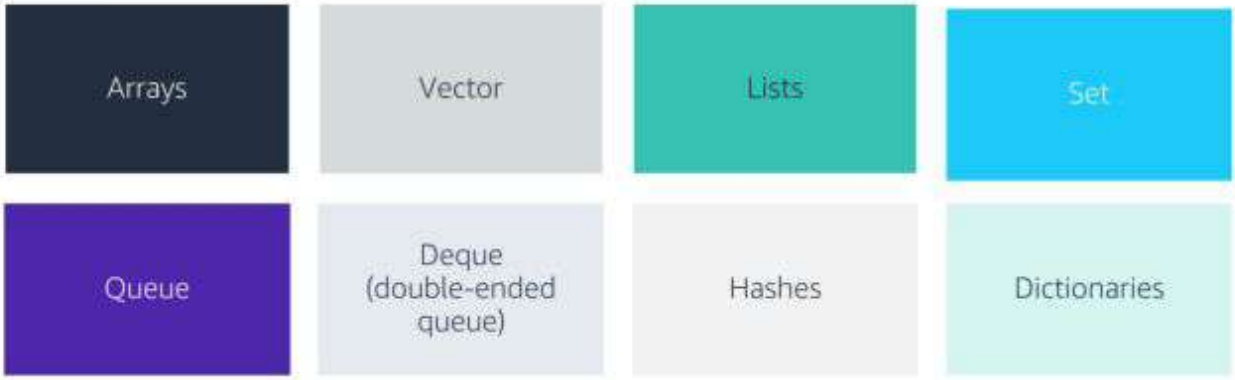
```
collisions =
predict(allAircraft)
```

```
latitude
longitude
speed
heading
verticalSpeed
windSpeed
windDirection
```

aws re/start

Collections

A *collection* groups multiple values in a single variable. Different types of collections are available:



The next few slides provides examples of collections.

Example collection: Array

The table shows a basic *array* of ages. Each age is assigned a slot, is of the same data type, and is adjacent to the previous age and the next age in memory.

- The first slot is almost always 0 in every language.
- Notice that the values do not need to be in any order.

Slot	Data Type	Value
0	Integer	87
1	Integer	10
2	Integer	2
3	Integer	46
4	Integer	22
5	Integer	19
6	Integer	66

Python collection example: List

A Python *List* is an **ordered** collection of items.

- Each item can be of a different data type.
- Items can be accessed using an *index*.
- Items can have duplicate values.

Index	Data Type	Value
0	String	"John"
1	Integer	55
2	Boolean	True
3	String	"male"
4	String	"Carlos"
5	Integer	22
6	String	"male"

A Python list is ordered because each new item is added to the end of the list.

- Follow the execution path of a program

What you will learn

At the core of the lesson

You will learn how to:

- Explain what an execution path is
- Describe the use of different types of flow control mechanisms, including loops, conditionals, and switches

aws re/start

In this lesson, you will learn how to:

- Explain what an execution path is
- Describe the use of different types of flow control mechanisms, including loops, conditionals, and switches

What does *execution path* mean?

- The sequence of steps that the program performs when it runs
- The program might …
  - Come to an either-or choice
  - Come to multiple choices
  - Perform work on each item in a loop
- The programmer must be able to predict what those steps will be…
  - When they write the code initially
  - When they debug problems that they encounter

Conditionals

- All programming languages have a way to choose an either-or path in the code.

```
if (guess > number):
    print("Too high!")
elif (guess < number):
    print("Too low!")
else
    print("Exactly right!")
```

Example from Python

```
if(name == 'Juan'):
    bonus = 300
else:
    bonus = salary * 0.1
```

Switches

- Most programming languages have a way to conveniently handle multiple possible cases of a value.

```
switch(sign):
    case "Stop":    pressBreak()
    case "Merge":   accelerate()
    case "Exit":    decelerate()
    default:            ignore()
```

Loops

- All programming languages have a way to do something on each item in a collection.

```
names = {“wei”, “nikki”, “akua”}

for name in names:
    newName = capitalize(name)
    print newName
```

Example from Python

```
for(employee in employees):
    employee.bonus = employee.salary * 0.1;
```

Version control

What you will learn

At the core of the lesson

You will learn how to:

- Explain the need for version control
- Explain the basics of Git
- Explain the difference between Git and GitHub

aws re/start

In this lesson, you will learn how to:

- Explain the need for version control
- Explain the basics of Git
- Explain the difference between Git and GitHub

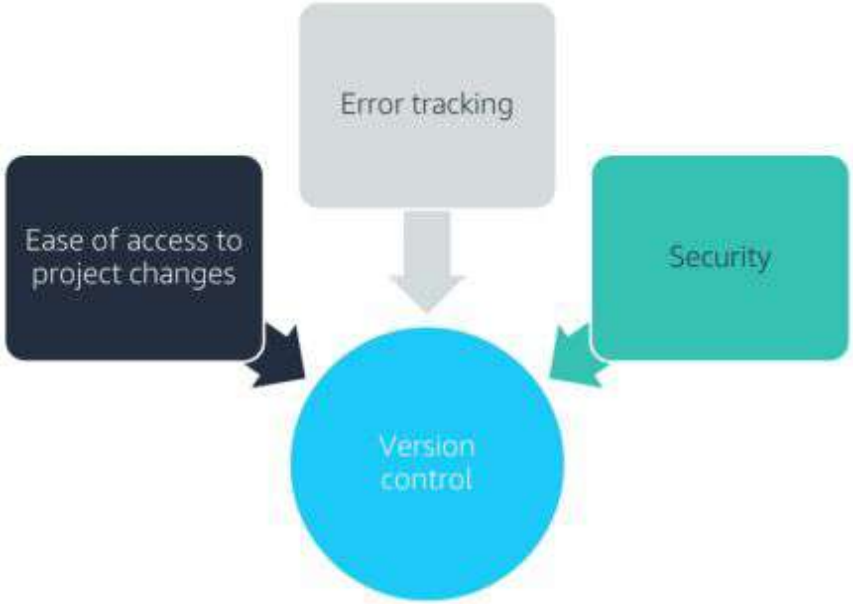
Version control and collaboration

- A version control system is software that tracks versions of your code and documents as you update them.

Version control can be done locally on your computer or by using a website that is dedicated to saving these versions.

Collaboration is doing version control, but in the cloud or on a dedicated website so that multiple people can work on a project.

Advantages of version control



aws re/start

Utilizing cloud infrastructure

The cloud, or a dedicated website, is useful for storing changes in code.

Version control that is only on a local computer can be easily lost, even if it is more secure than saving multiple versions of a file.

Data that is stored in the cloud has a reduced risk of being lost.

Version control tools

Several version control tools are available:

- Git and GitHub
- GNU arch
- Mercurial

More version control tools exist, but the one that this course focuses on is Git and GitHub.

Mercurial  GitHub

Git

- Install the Git tools
- Add files to a repository
- Edit files in a Git repository
- Clone an existing repository
- Create and merge branches
- Rewrite history in a Git repository
- Resolve merge conflicts

Git, continued

The best way to learn Git is to experience it.

1. Install the Git Tools from the website: About Git.
2. After you install the Git Tools, open the Git Bash program.
3. In the Bash terminal, create a folder or directory that is named *GitTest* by entering: *mkdir GitTest*
4. Change directories by entering: *cd GitTest*
5. Create a local Git repository by entering: *git init*
6. Confirm that a new repository is there, by entering *ls -ldF .** and looking for the .git repository.
7. Next, create a new file by entering: *touch newFile.txt*
8. Open the new file by entering: *nano newFile.txt*

Git, continued

9. Enter any text that you want.
10. To exit from nano, press CTRL+X.
11. In the terminal, enter: *git status*
    You should see newFile.txt in red characters.
12. Now, add newFile.txt to the local Git repository by entering *git add newFile.txt* and then *git status*
    You should see that newFile.txt is in now in green characters, which means that it is ready to be committed.
13. To finish the process, enter *git commit -m "Initial commit"* followed by *git status* to confirm that Git is not tracking any new changes.

Git, continued

If the commit message did not work, you most likely saw a message that said *Please tell me who you are* in the terminal.

14. To clear this message, in the terminal, enter the following two commands :
    - *git config –global user.email "<you@example.com>"*
        - Replace <you@example.com> with an email address that you actually own and want to use for this course.
        - The rest of the command is entered exactly, including the quotation marks (" ").
    - *git config –global user.name "<Your name>"*
        - Replace <*Your name*> with a user name that you want to use for this course. Make it unique.
        - The rest of the command is entered exactly, including the quotation marks.

Git, continued

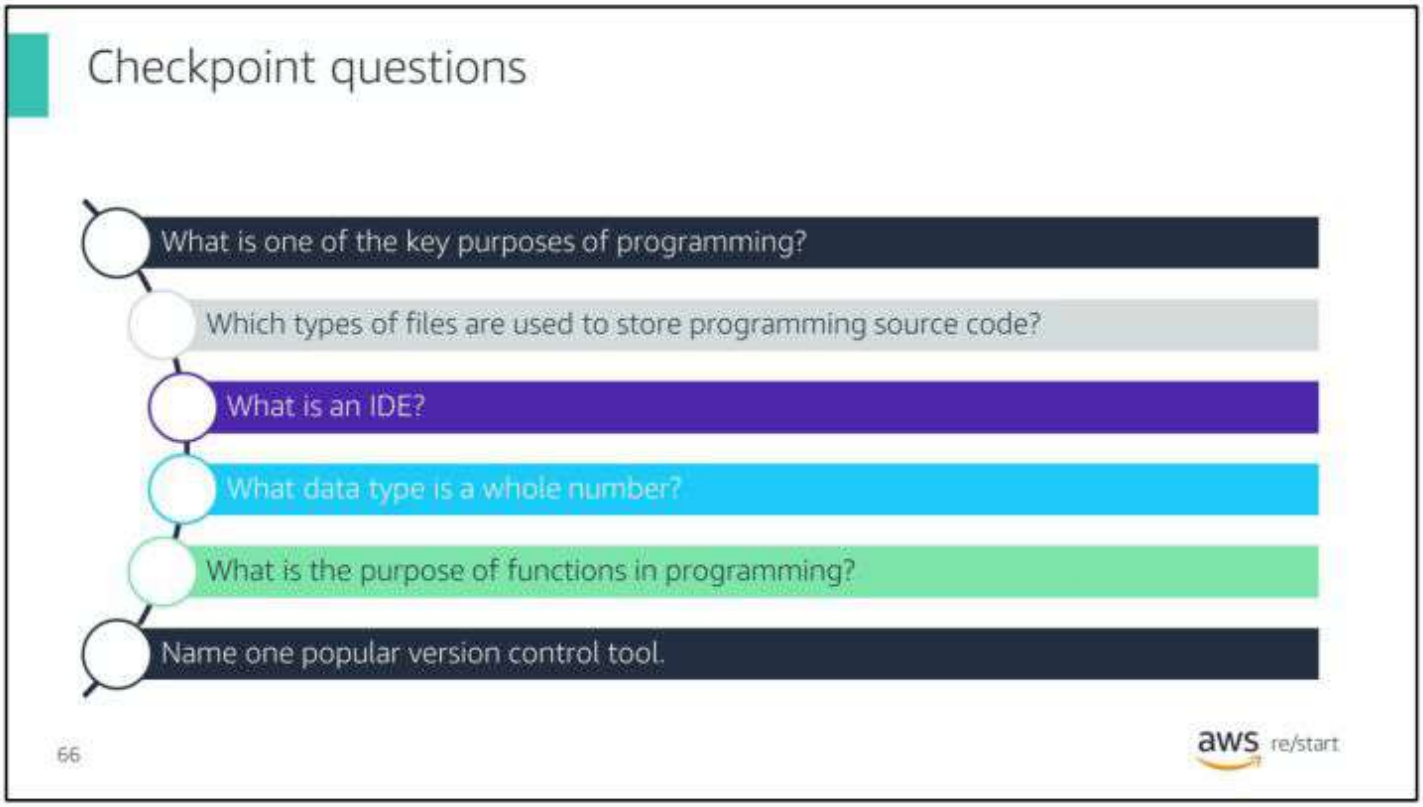
15. After you are finished, enter *git commit -m "Initial commit"* again, followed by *git status* to confirm that Git is not tracking any new changes.

GitHub

GitHub is a repository hosting service. It uses repositories and the same commands as the terminal.

To prepare for the lab, create a GitHub account on GitHub, if you don't already have one.

Sites like GitHub are also a good way to exhibit your code to developers and potential employers.



Answers:

1. From the lesson, it was shown that automation is one of the uses of programming.
2. Programs are written in text files.
3. An IDE is an integrated development environment that helps with writing code.
4. Whole numbers are typically stored as integers.
5. Functions enable developers to create discrete sections of code that can be called by using the function's name. The resulting function is then available to the rest of the source code.
6. Git.

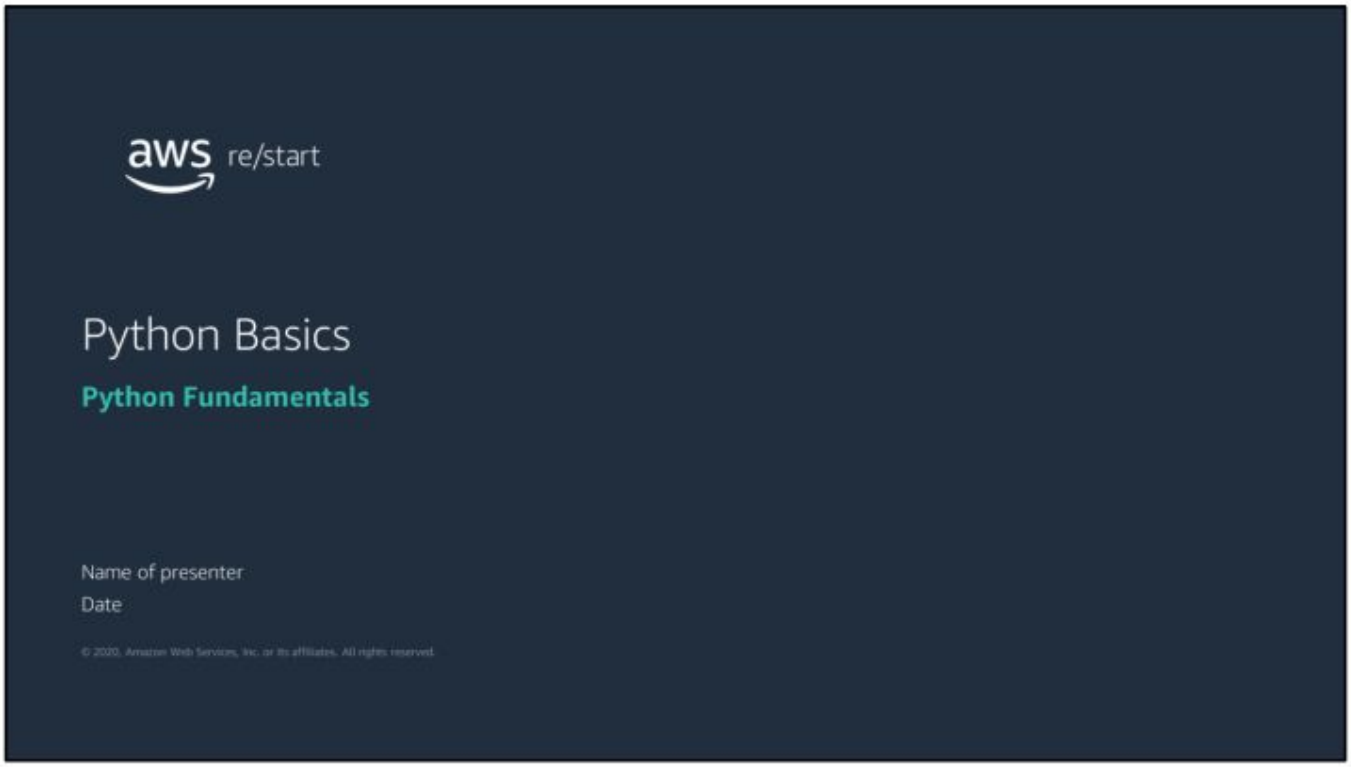
Key takeaways

- Programming is a way to automate processes.
- Programming languages specify a way to communicate directions to a computer.
- Software is written into a text file by using a programming language, which is either interpreted or compiled when it is run.
- Data is *typed* so that the interpreter or compiler knows whether it is a string, integer, Boolean, or other data type.
- A composite data type stores different types of data in a single variable.
- Functions are collections of instructions that can be called repeatedly in a program.
- Version control manages changes to computer programs, documents, or other collections of information.

aws re/start

Some key takeaways from this lesson include:

- Programming is a way to automate processes.
- Programming languages specify a way to communicate directions to a computer.
- Software is written into a text file by using a programming language, which is either interpreted or compiled when it is run.
- Data is *typed* so that the interpreter or compiler knows whether it is a string, integer, Boolean, or other data type.
- A composite data type stores different types of data in a single variable.
- Functions are collections of instructions that can be called repeatedly in a program.
- Version control manages changes to computer programs, documents, or other collections of information.



Welcome to Python Basics.

What you will learn

At the core of the lesson

You will learn how to:

- Install Python on a Linux computer
- Define basic Python terminology
- Declare variables in a Python script and perform operations on those variables
- Explain statements, functions, and exceptions

aws re/start

In this module, you will learn how to:

- Install Python on a Linux computer
- Define basic Python terminology
- Declare variables in a Python script and perform operations on those variables
- Explain statements, functions, and exceptions

System requirements for Python

Microsoft Windows

macOS

Linux

Demo: Install Python

Check the current installation of Python and install a newer version of Python.

4

aws re/start

1. By default, CentOS includes Python V2.7.
2. You can check which version you have by using the following command:
   `# python --version`
3. You can observe that Python V2.7 is running. (Note: For the purpose of this course, you will use Python V3.7. You will learn how to update Python versions.)
4. Run the following command:
   `# yum install gcc openssl-devel bzip2-devel libffi-devel`
   (This command installs a series of packages. At the end, the installer will ask if it is ok. Respond by entering: *y*)
5. Run the following commands:
   `# cd/usr/src`
   `# wget https://www.python.org/ftp/python/3.7.2/Python 3.7.2.tgz`
   (The second command tells CentOS to *get* the specific file from ***python.org***.)
6. Run the following command:
   `# tar xzf Python-3.7.2.tgz`
   (This command tells CentOS to extract the file.)
7. Run the following command:
   `# cd Python-3.7.2 # ./configure --enable-optimizations # make altinstall`
   These commands take some time to run.

8. Now, remove the downloaded file from the system:
   `# rm /usr/src/Python-3.7.2.tgz`
9. You will be asked if you want to remove this file. Enter ***y*** and press **ENTER**.
10. Verify the installation by running this command:
    `# python3.7 -V`

Python syntax basics

Python uses indentation and spacing to group blocks of code together.

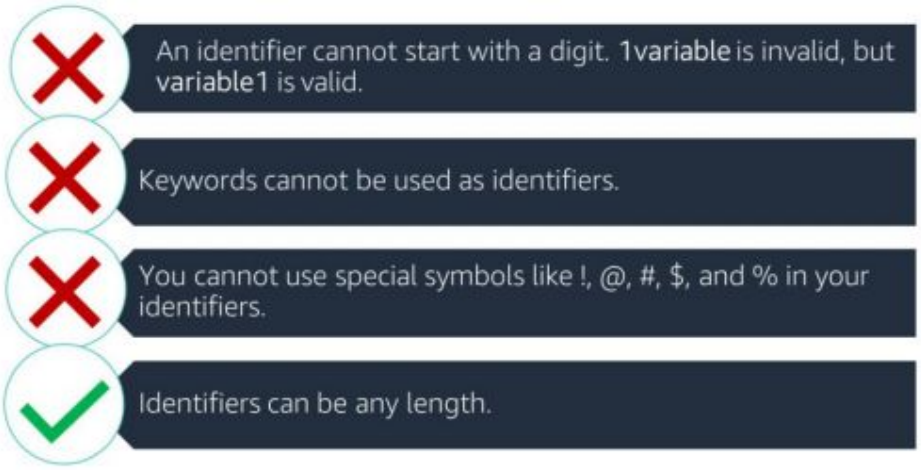
- If you get runtime errors, check your spacing first.
- Next, check your indentation for any missing punctuation, such as colons.

Python is also case sensitive. Capitalization matters.

Identifiers

In Python, *identifier* is the name for entities like class, functions, and variables. It helps differentiate one entity from another entity.

When you name objects in Python, you must observe some rules:



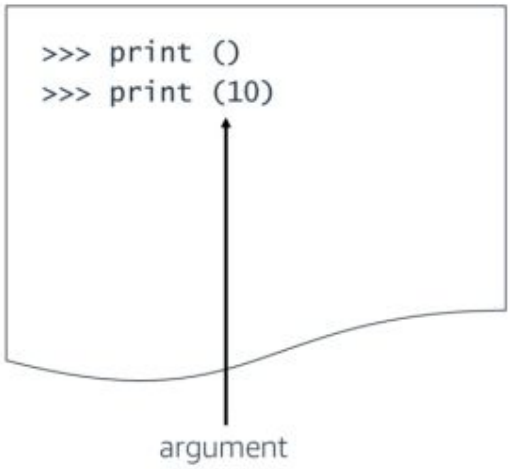
aws re/start

File extensions: .py files

The extension *.py* is a script file extension that is used in Python.

Files with the .py extension can be created in any text editor, but they need a Python interpreter to be run.

Functions



- A function tells the computer to do a specific task.
- Functions have a *name*, which is called by adding parentheses after the name.
- Many functions take *arguments*, which are information that the function uses to do its specific task.
- One useful function is the **print** function.
- You can write your own functions.
- Python also has native functions (such as **print**) that are already built in.

Vim review

The text editor that you will use is Vim (Vi improved).

- To open it on the command line, enter *vim <file name>* or *vim*
- Vim opens in normal mode. To write to Vim, enter *i* (the *i* stands for *insert*)
- To get back to normal mode, press ESC
- When you are in normal mode, you can save and quit, or continue editing
- To save a file, enter *:w <file name>* or *:w* (the *w* stands for *write*)
- To close Vim, enter *:q* (the *q* stands for *quit*)
- To save and close, enter *:wq*

Demo: Hello World

1. Create a file in CentOS that is called *helloworld.py*.
2. Open that file and write to it:
   print("Hello World")
3. Save the file and return to your command line.
4. Run the file with Python and observe what happens.

aws re/start

1. Enter the following command:
   `# vim helloworld.py`
   (This command creates the file. Enter *i* to insert text.)
2. Enter ***print("Hello World")*** and press ENTER.
3. Press ESC, then enter: ***:w***
   (*:w* writes the file.)
4. Then enter: ***:q***
   (*:q* quits and puts you back on the command line.)
5. At the command line, enter this command:
   `# python3.7 helloworld.py`
   (This command runs the file with the Python interpreter.)

Comments

```
# This is a comment
```

- Comments are notes to yourself and to other developers.
- Comments describe the contents of a program and how it works so that a person who reviews the source code can understand it.
- Use the pound (#) symbol to start writing a comment.
- By starting a line of text with #, you tell the interpreter *not* to run this line as code.

Demo: Hello World and Commenting

1. Open your *helloworld.py* file.
2. Insert more text by entering *i*
3. On another line, enter some comments.
   - For example: `# I love Python`
4. Write and exit.
5. Run your file again by using the following command:
   `python3.7 helloworld.py`
6. Did anything change? Why or why not?

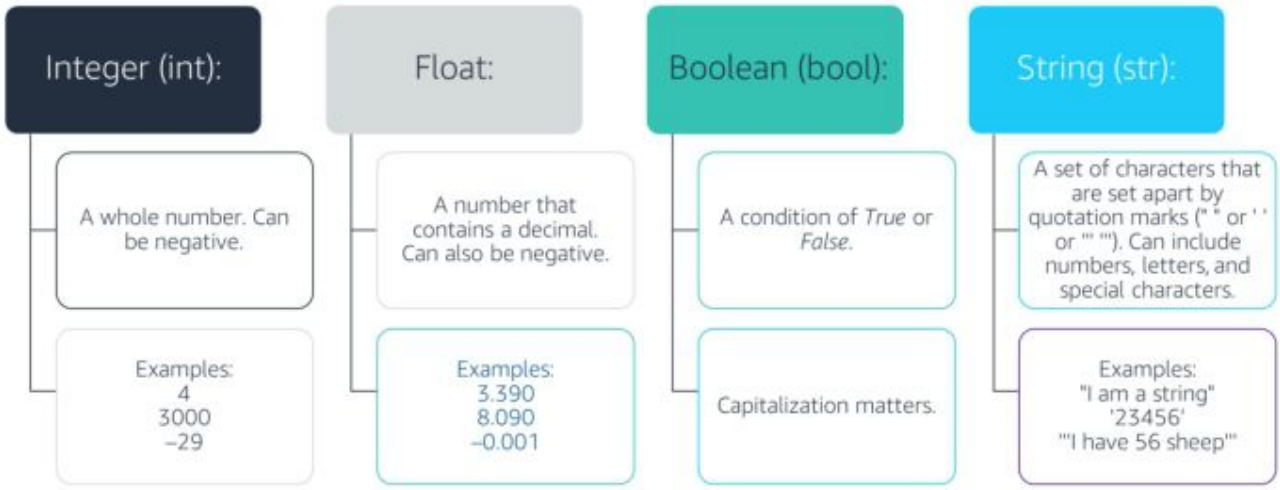
Data types

Data types determine whether an object can do something, or whether it makes sense.

Python stores the type of an object with the object. When the operation is performed, it checks whether that operation makes sense for that object. (This technique is called *dynamic typing*.)

Example: Some functions or methods in Python expect a certain data type. You cannot pass a string into a function that is expecting a float.

Basic data types



Note: Numbers inside a string are still a string. Python does not consider them to be integers or floats.

Data types: Mutable versus immutable

Mutable means *changeable*. In Python, some data types are *mutable*, and others are *immutable*.

Converting data types

Python has specific commands that you can use to change an object from one data type to another.

- **float()**: In the example, *4* is assigned to the variable *x*. By its nature, *4* is an integer. However, *x* must be identified as a float, so the **float()** command was used. The result is *4.0*.
- **int()**: By passing *x* back into the **int(x)** command, it receives *4* back. This command truncates the decimals.
- **Note**: Using **int()** is not the same as rounding. It only removes any decimal digits that the number has. It does not round the decimal numbers up or down, so be careful.

Lists, sets, tuples, and other data types will be covered later.

Strings

```
>>> "I'm a string!"
>>> 'Me too'
>>> '''Me too!'''
>>> "I have written 3 strings"
```

- Strings contain *text*, which can be a single character or paragraphs of text.
- Strings are characters that are enclosed between quotation marks.
- Strings can contain numbers.
- Three ways of notating a string are:
  - Single quotation marks (' ')
  - Double quotation marks (" ")
  - Triple quotation marks (''' ''')

String concatenation

```
>>> "I'm a string!"
>>> 'Me too'
>>> "I'm a string" + "Me too"
>>> "I'm a stringMe too"
```

Note: If you want a space between the words of your new string, you must add it into one of the original strings. Otherwise, Python adds them as-is, without inserting a space.

- Strings are *immutable*. When you manipulate them, you create a new string.
- You can *concatenate* or *add* strings together, which creates a new string.
- It is possible to create an empty string.
  - Example: **x = ""**

A simple Python program

```
>>> 1 + 1
>>> 2
```

- This simple Python program adds the numbers *1* and *1* together.
- The result is *2*, as displayed on the following line.
- The angle brackets (>>>) are not part of the program. The Python interpreter displays them.

Variables

```
>>> x = 1
>>> x + x
2
```

- In this example, *x* is a variable that holds the number *1*.
- When you do something with variables, Python looks up the values to understand what to do.

Variables cont.

```
>>> apples = 2
>>> oranges = 3
>>> apples = oranges
>>> car_color = red
```

- You can have as many variables as you want.
- You can name your variables whatever you want, within certain restrictions.
- Restrictions are:
  - A variable name can only contain letters, numbers, and underscores (_).
  - A variable name cannot begin with a number.
  - A variable name cannot be a keyword in Python.

Demo: Variables

22

Instructions

1. Create a new file that is called: *variables.py*
2. Make some variable assignments of your choice. Use strings, floats, or ints.
3. Examples:
```
x = 33
y = 7.0
variable_challenge = "I love Python"
```
4. Use the **print()** function to tell Python to print your variables. Otherwise, when you call the file, you will not see any output.

   Examples:
```
print(x)
print(y)
print(variable_challenge)
```
5. Write and exit your file. Call the file from the command line. What happened?

aws re/start

Example code:

```
x = 33
y = 7.0
variable_challenge = "I love Python"

print(x)
print(y)
print(variable_challenge)
```

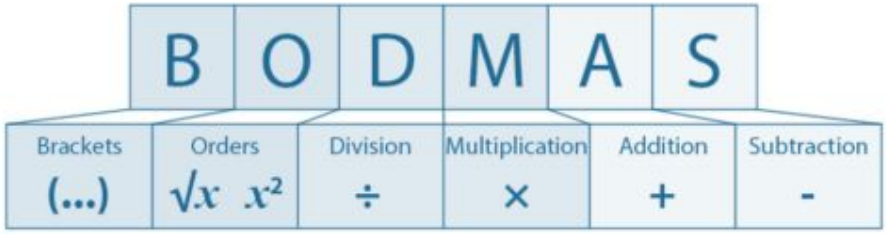
Operators

Arithmetic operators	(+ - * / % ** //)
Comparison (relational) operators	(<, >, <=, >=, !=)
Assignment operators	(=, +=, -=, *=, /=, %=)
Logical operators	(and, or, not)
Membership operators	(in, not in)
Identity operators	(is, is not)

BODMAS

BODMAS explains the order of operations:

- Bracket
- Orders (exponentiation)
- Division
- Multiplication
- Addition
- Subtraction



Demo: Basic Math

25

Instructions

1. Create a new file that is called: *basicmath.py*
2. Insert text into the file. Try writing some basic math functions.
   Examples:
   `1+1`
   `100/33`
   `2**2`
3. Now, tell Python to print these expressions by using the **print()** function.
4. Write and exit the file.
5. Run it. What happened?

aws re/start

Create a new file with the following code:

```
print(1 + 1)
print(100/33)
print(2**2)
```

Demo: Variables + Basic Math

Instructions

What happens when you combine variable assignment with math?

1. Open your *basicmath.py* file.
2. Edit the file to assign variable names to your basic math functions.
   Examples:
   ```
   easy_math = 1+1
   one_third = 100/33
   powerful = 2**2
   ```
3. Tell Python to print those variable names.
4. You can also do math with only the variable names.
   Examples:
   ```
   x = 2
   y = 4
   z = x+y
   print(x + y)
   print (z)
   ```
5. Do you expect printing *z* and *x+y* to display the same result? Why or why not?

Demo: Strings and Concatenation

Instructions

1. Create a new file: *funwithstrings.py*
2. Create a variable and assign a string to it.
   Example:
   `first_str = "I love dogs"`
3. Create a second variable and assign a different string to it.
   Example:
   `second_str = " and cats"`
4. Create a third variable that is called *newstr* and assign it the concatenation of strings 1 and 2.
   Example:
   `newstr = first_str + second_str`
5. Print *newstr*. What happened?

aws re/start

Statements

```
>>> apples = 2
>>> oranges = 3
>>> apples = oranges
```

- A *statement* is usually a line of code, and each line of code that you have seen so far is a single statement.
- A statement is an individual instruction to Python.
- This code has three statements.
- By assigning *apples* to *oranges*, you are changing the value of *apples*. If you `print(apples)`, what is the response?

Functions, strings, and variables

```
>>> name = "Kwesi"
>>> print (name)
```

- Store strings in variables.

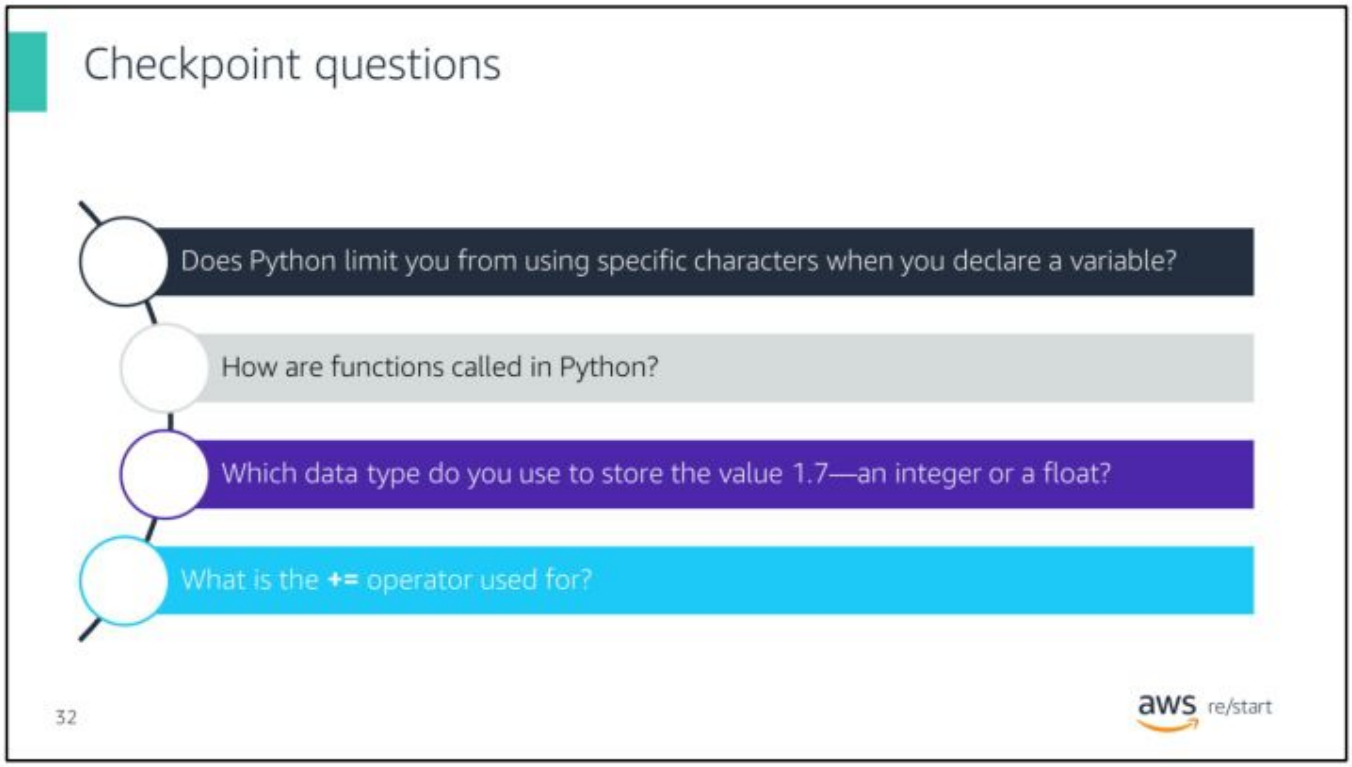
Exceptions

- An exception raises an error.
- Examples include:
  - Referencing a variable by the incorrect name
  - Dividing a number by zero
  - Trying to read a file that does not exist on your computer
- Exceptions result in a *stack trace*, which is a listing of the various ways that something went wrong.

Demo: Find the Exceptions

Instructions

- Do some research online for Python exceptions. Find as many different kinds of exceptions as you can. Be prepared to discuss your findings with the class.

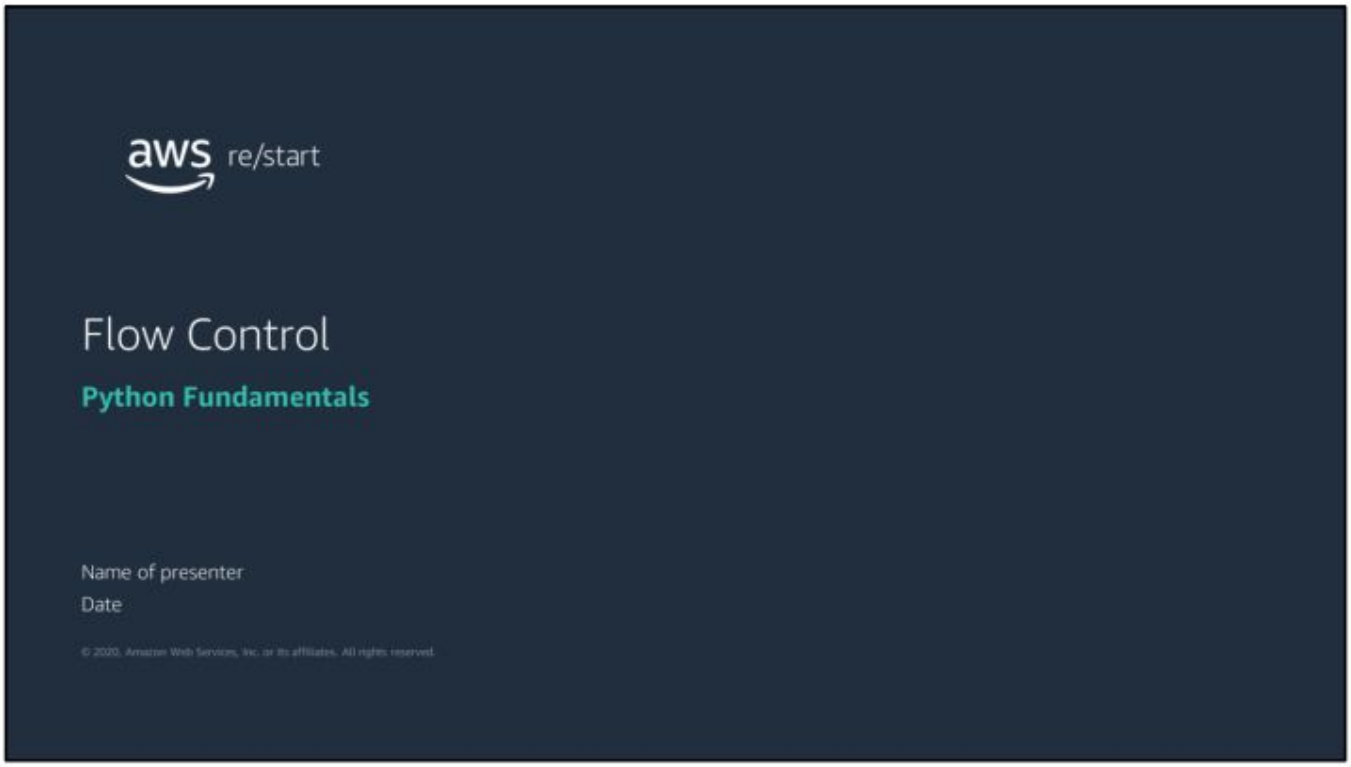


Answers:

1. Yes, you cannot use symbols such as !, @, #, $, and others in variable declarations.
2. Functions are called by name.
3. A float is used to store decimal values.
4. The plus equals (+=) operator adds the *rvalue* (the value to the right of the operator) to the *lvalue* (the value to the left of the operator). The lvalue is usually a variable. For example, if *x* is *5*, consider the following code:
   x+=10
   After that code is run, the value of *x* is *15*.

Some key takeaways from this lesson include:

- Python can be installed on Microsoft Windows, macOS, and Linux computers.
- Python files have the *.py* extension.
- Python used indentation and spaces, and is case sensitive.
- Functions are collections of commands that can be called by name.
- Data types identify the type of data (integer, float, and so on) that a variable contains.
- Operators in Python are used for math, equivalence tests, and string operations.
- Exceptions are errors that the Python interpreter raises when an error occurs in the code.



Welcome to Flow Control.

What you will learn

At the core of the lesson

You will learn how to:

- Describe the purpose and use of flow control in a program
- Use conditional statements to run code based on evaluated criteria
- Use loops to run code repeatedly until the criteria are met
- Use the input() function to prompt users for input to be used when they run the program

aws re/start

In this module, you will learn how to:

- Describe the purpose and use of flow control in a program
- Use conditional statements to run code based on evaluated criteria
- Use loops to run code repeatedly until the criteria are met

Flow control

Flow control is the order that individual statements, instructions, or function calls of an imperative program are run or evaluated.

If: The keyword ***if*** means that if this condition is met, complete this action.

Elif: The keword ***elif*** is short for *else if*. Else, if this condition is true, complete this other action. (You can have multiple ***elif*** statements in a single block.)

Else: The keyword ***else*** means that if none of the above conditions are met, complete this action.

aws re/start

Conditional statements: Examples

Some examples of how conditionals work in your everyday life are:

If the time is between these hours, the store is open. Else, the store is closed.

If the gas in the gas tank is below a certain level, turn on the low fuel warning.

If an employee has their badge, allow them into the building. Else, the door is locked.

If someone's age is greater than 18, consider them to be an adult. Else, if their age is between 13 and 18, consider them to be a teen. Else, consider them a child.

Loops

- Loops are a technique that tells Python to run a block of code repeatedly.
- Python runs loops *for* a certain number of times, or *while* a certain condition is true.
- The two types of Python loops are called **for** and **while**.

Demonstration: Conditionals

1. Create a new file that is called: **conditionals.py**
2. Create a set of conditionals using "if", "elif", and "else" statements in Python. They should be created so that:
   - *If there are 5 or more bananas print "I have a bunch of bananas."*
   - *if there are 1–4 bananas print "I have a small bunch of bananas."*
   - *If there are no bananas print "I have no bananas."*
3. Assign **print** statements to your conditions using the above rules.
4. Create a variable to hold the number of bananas.
5. Run the code to observe the result.

aws re/start

7

Code example:

```
bananas=1

if bananas >= 5:
        print("I have a large bunch of bananas")
elif bananas >= 1:
        print("I have a small bunch of bananas")
else:
        print("I don't have any bananas")
```

Conditionals in code

Consider the previous example of human age and life stages, and code it into Python. With conditionals, the indentation becomes important. You must group or nest the related conditionals together.

```
if age > 18:
...         print("adult")
else:
...         print("child")
```

You must tell Python which age you want it to evaluate. Create a variable that is called *age* and assign it a numerical value. You must do this step *before* you write your conditional statements.

Demonstration: While Loops

1. Create a new file that is called: **while.py**
2. Create a **while** loop with the code from the previous slide.
3. Assign **print** statements to your loop.
4. Run the code and observe the result.

While loops

- **While** loops can run indefinitely, so you must include the condition for the loop to stop. Otherwise, the code creates an infinite loop.
- It is common to use an *iterative counter* with loops. As the loop completes, the counter increases (or decreases). When the counter reaches a specific number, the loop stops.

Example:

```
counter = 0

while counter <= 3:
        print("I love learning Python!")
        counter = counter + 1
```

Conditionals in code, continued

Your code might look like this example:

```
age = 21

if age >18:
        print("adult")
else:
        print("child")
```

Some things to note:

- The colons *after* the conditionals are important.
- You also use the **print** function. If you run this code, what do you think Python will print?

Lists

```
>>> [1, 2, 3]
>>> ["alice", "bob"]
>>> [1, "alice", 2, "bob"]
>>> []
```

- Lists are a mutable data type. Lists can contain multiple data types (strings, ints, floats, and even other lists).
- Lists are denoted with brackets ([ ]) on each end.
- Values are enclosed in brackets, and they are separated with commas.
- Any number of items can be in a list—even zero (no) items.

Demonstration: Lists

1. Create a file that is called: **lists.py**
2. Create a list, and place three or more items in that list. Try mixing the data types, or making multiple lists.
3. Print your list.

aws re/start

Code demo:

```
class_roster = [“Xiulan","“Kwaku","“Shirley"]
test_scores = [86,93,80]

print(class_roster)
print(test_scores)
```

For loops

- A **for** loop reads: for each element in ***<thing>***, do a certain task.
- Some real-life examples are:

For every egg in a recipe, add 2 cups of flour.

For every package on our truck, add 2 kilograms.

For every hamburger that is ordered, subtract 1 from the inventory.

Loops and lists

```
for num in [1, 2, 3]:
    ...
    print(num)
```

- **For** loops and lists work well together.
- For every item in this list, Python prints that item.
  - Every time it calls **num**, it assigns a value from the list (1, then 2, . . .) to **num**.
  - The loop then prints the value.
  - After it goes through the entire list of values, the loop stops.

Demonstration: For Loops

1. Open your **lists.py** file.
2. Create a **for** loop that prints your list.
3. Run the code and observe the result.

aws re/start

Code demo:

```
class_roster = [“Xiulan","Kwaku","Shirley"]
test_scores = [86,93,80]

for student in class_roster:
        print(student)

for score in test_scores:
        print(score)
```

Input

```
>>> response =
input("What's your name? ")
What's your name? Jorge
>>> response
'Jorge'
```

- The **input()** function asks the user to enter text and saves the result to a variable.
- One optional argument is a prompt for the user, as a string.

Dictionaries

```
>>> {"key": "value"}
>>> {0:100, 1:200, 2:999}
>>> {0:{1:2}}
>>> myDict = {}
```

- Dictionaries contain immutable keys, which are associated to their values. Keys must be immutable data types.
- Dictionaries can be nested inside each other.
- To create an empty dictionary, use a pair of braces with nothing inside: `{}`
- Keys are separated from their values with a colon: `{"Key":"Value"}`
- Retrieve a value in the dictionary by its key: `myDict.get("key") or myDict["key"]`

Demonstration: Dictionaries

1. Create a file that is called **dictionary.py**.
2. Create a dictionary and create four ***key:value*** pairs in it.
3. Print the third value in your dictionary.
4. Print your entire dictionary.

aws re/start

Code example:

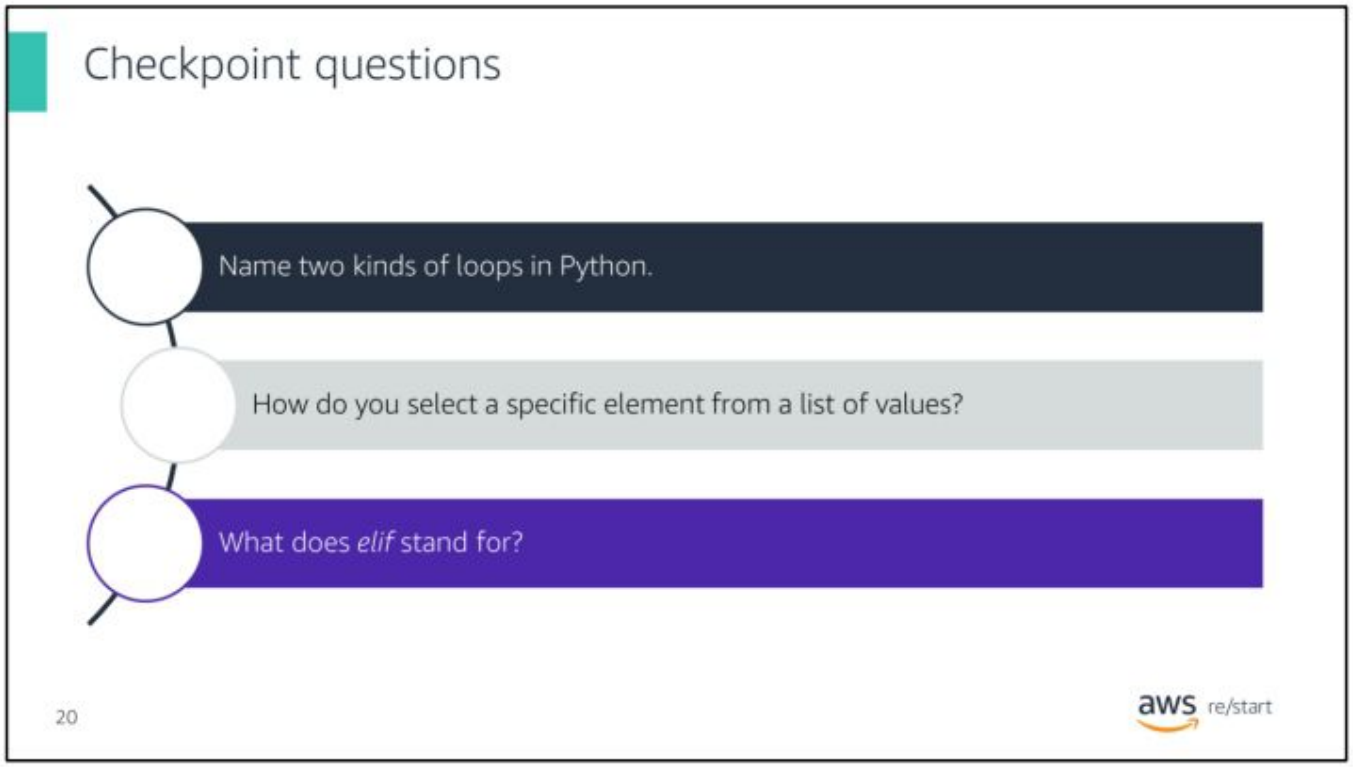
```
myDict = {}
myDict["one"] = 1
myDict["two"] = 2
myDict[3] = "three"
myDict["four"] = 4.4

print(myDict[3])

for key, value in myDict.items():
        print(key, value)
```

Demonstration: "Hello, Your Name"

1. Open **helloworld.py**.
2. Get input from user.
3. Print the response.
4. Can you concatenate the input so that the output prints **hello *<response>***?
5. Run the program again.

Code example:

```
name = input("What's your name?")
print("Hello " + name)
```



Answers:

1. Two kinds of loop flow structures in Python are *for* and *while*.
2. Specific values in a list are specified by an index in brackets ([ ]) after the list name. For example, *myList[2]* selects the third element in the *myList* variable.
3. The statement *elif* follows an *if* statement. If the preceding *if* or *elif* conditional statement evaluates to false, then the *elif* is evaluated.

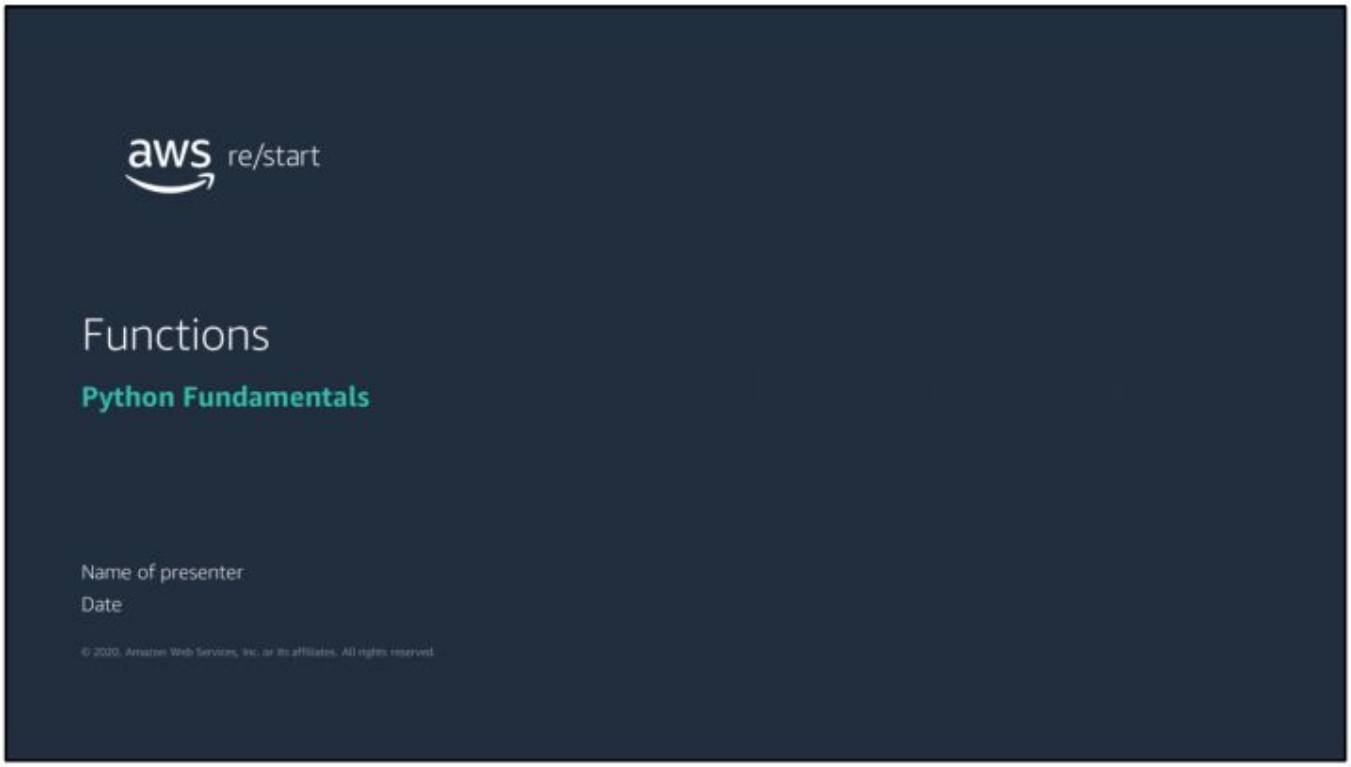
Key takeaways

- An *if* conditional statement evaluates a condition and, if *true*, it runs a block of code.
  - *elif* and *else* statements follow *if* statements
- A *while* loop runs a code block until a condition returns *false*.
- A list is a collection of values in a specific order. The values do not have to be the same type.
- A dictionary is a collection of key-value pairs, where the value is accessed by using the key.

aws re/start

Some key takeaways from this lesson include:

- An *if* conditional statement evaluates a condition and, if *true*, runs a block of code.
  - *elif* and *else* statements follow *if* statements
- A *while* loop runs a code block until a condition returns *false*.
- A list is a collection of values in a specific order. The values do not have to be the same type.
- A dictionary is a collection of key-value pairs, where the value is accessed by using the key.



Welcome to Functions.

What you will learn

At the core of the lesson

You will learn how to:

- Explain the purpose of functions
- Name the different types of functions
- Use functions to organize Python code

aws re/start

In this module, you will learn how to:

- Explain the purpose of functions
- Name the different types of functions
- Use functions to organize Python code

Functions

In Python, a function is a *named sequence of statements that belong together.*

Their primary purpose is to help organize programs into chunks that match how you think about the solution to the problem.

First, define the function. Name it and put placeholders for arguments. Indent lines of code inside the function, like code in loops.

Example:

```
def <function name>(argument):
    <things to do>
```

Functions, continued

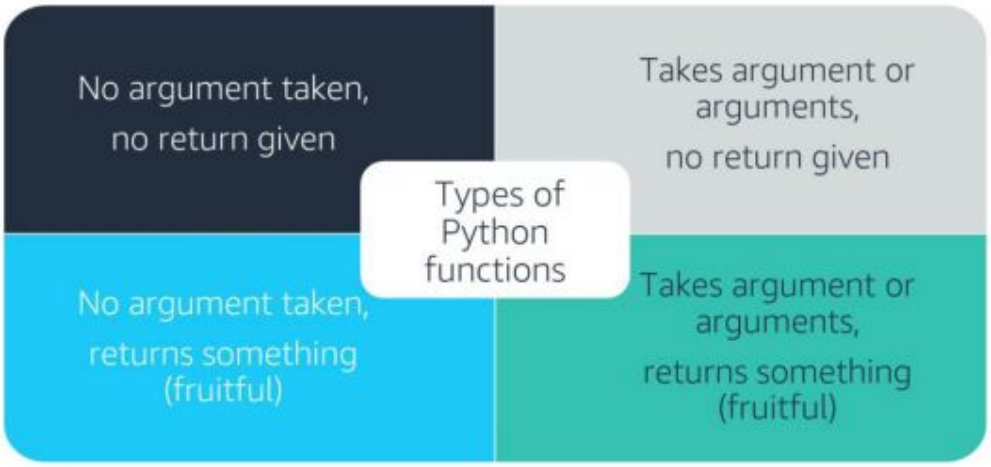
Functions are used when you must use the same block of code several times. Reusability is the primary reason for functions.

Built-in functions (like **print**) are part of a programming language, or developers can create new functions.

Python has many built-in functions that solve many common problems for you.

Examples:
- `print()`
- `open()`
- `sum()`
- `dict()`
- and others

Types of functions



Example function 1

```
def demo(x):
        y = x + 3
        return y

print(demo(3))
```

- What is the identifier (name of this function)?
- What is the argument?
- What is it returning?
- Is this function fruitful or non-fruitful?

On the print line, multiple things are happening:

- The function **demo** is being called. Functions do not run until you call them.
- The **print** function is also being called, and it uses ***demo*** as the argument for `print()`. It can be helpful to call a function with another function.

- What output do you expect to be printed?

Example function 2

```
a = 3
b = 2
c = 1

def demo():
        y =(a+b+c)

demo
~
~
```

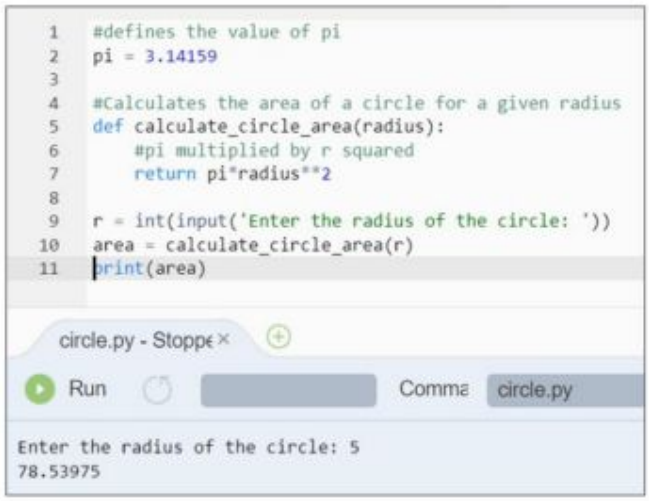
- Is this function fruitful or non-fruitful?
- When you run this function, what do you expect to happen?
- What might be some of the advantages of using an argument in a function?

Organizing code with functions

Organizing code with functions makes it easier to read.

Example:

- It can be difficult to interpret what the first line of code does.
- Using an appropriately named function that takes appropriately named arguments makes the code easier to interpret and use.



aws re/start

The **input** function prompts a message and waits for the user to enter a value (5 in this example)
This value is converted into an integer by the **int(...)** function and stored in the variable **r**
The function **calculate_area_circle** is called with **r** as a parameter. It returns the area of a circle for a radius of value **r**
The value returned by **calculate_area_circle** is stored in the variable **area**
The **print** function displays the value of the variable **area** in the console

Demonstration: Use Functions to Organize and Reuse Code

1. Write a simple function.
2. Run a sample program.
3. Review the results.
4. Define and use functions without arguments.
5. Define and use functions with arguments.
6. Compare outputs.

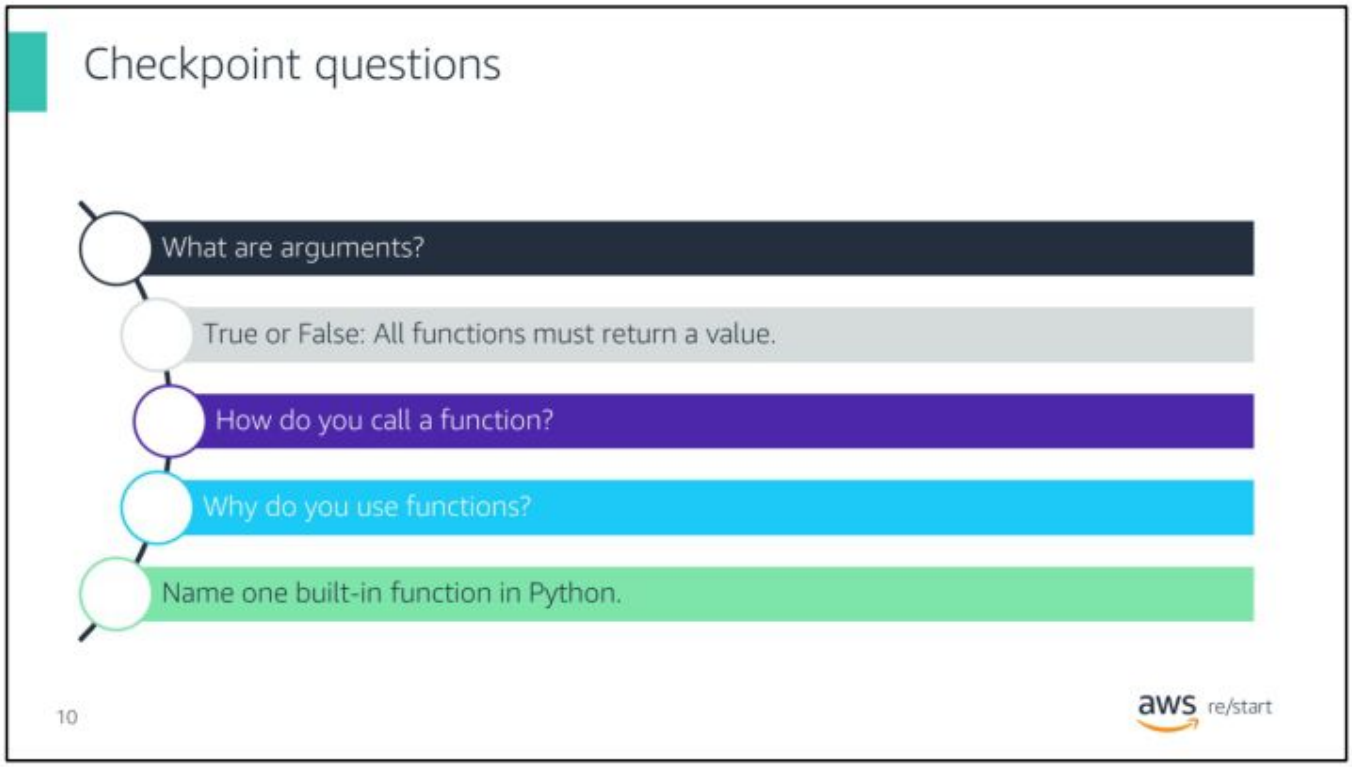
aws re/start

Sample code:
Items 1, 2, 3, & 4

```
def greet_user():
    print("Hello there!")


greet_user()
```

Item 5

```
def greet_user(name):
    print("Hello " + name)

greet_user("Sam")
```



Answers:

1. Function arguments enable developers to pass values to a function. For example, a function that is called *setColor* could include a string for the color that is being passed. Such a call would appear as *setColor("red")*.
2. False.
3. You use the name of the function and then its argument in parentheses (which might be empty). For example: *setColor("red")*, ***clearScreen()***
4. Functions enable developers to use the same code many times without retyping the statements.
5. One of the most commonly used built-in functions is print.

Key takeaways

- Functions are used when you must perform the same task multiple times in a program.
- Functions are called by name and the function call often includes arguments that the function code needs for processing.
- Python includes many built-in functions, such as **print** and **help**.

aws re/start

Some key takeaways from this lesson include:

- Functions are used when you must perform the same task multiple times in a program.
- Functions are called by name and often include arguments that the code in the function needs for processing.
- Python includes many built-in functions such as print and help.

Welcome to Modules and Libraries.

What you will learn

At the core of the lesson

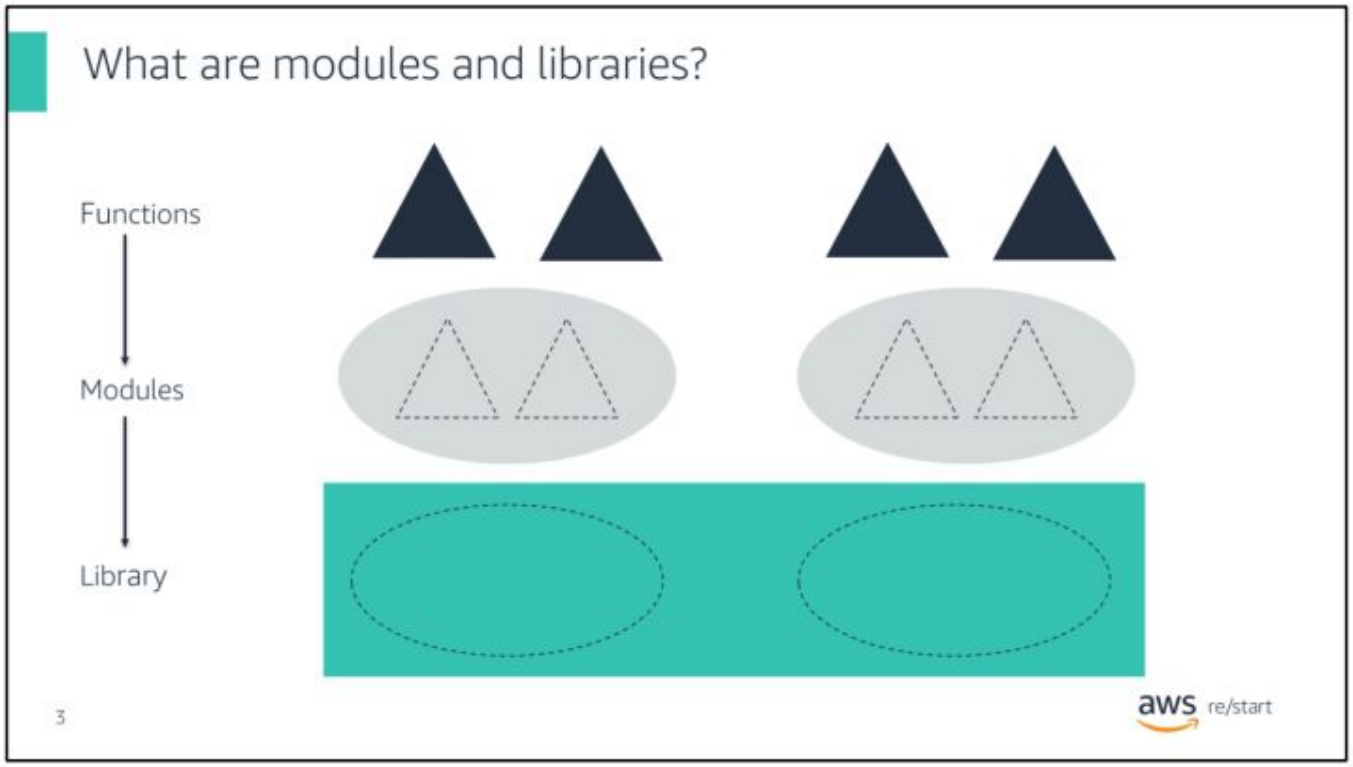
You will learn how to:

- Explain the purpose of a module in Python
- Describe the Python standard library
- Import modules from a library
- Use file handlers to open, read, write, and close files from within Python code
- Use exception handlers to catch errors in code
- Import the JavaScript Object Notation (JSON) module and use JSON functions
- Use pip to download and install third-party modules for Python

aws re/start

In this module, you will learn how to:

- Explain the purpose of a module in Python
- Describe the Python standard library
- Import modules from a library
- Use file handlers to open, read, write, and close files from within Python code
- Use exception handlers to catch errors in code
- Import the JavaScript Object Notation (JSON) module and use JSON functions
- Use pip to download and install third-party modules for Python



Functions are blocks of code that do specific tasks.

Functions can be put into modules. Modules are separate Python files that can be imported into other Python applications. Importing modules makes it possible to reuse code.

Libraries are collections of modules. By putting modules into libraries, it is possible to import a large amount of programming capability quickly.

Standard library

The Python standard library is a collection of script modules that a Python program can access. It simplifies the programming process and reduces the need to rewrite commonly used commands.

Python libraries can also consist of modules that are written in C.

Practical example: Saanvi Sarkar did not invent the wheel. Instead, she imported a wheel module from a library to help create something new—a car.

Navigating the Python standard library

The Python standard library is a collection of modules that make programming easier by removing the need to rewrite commonly used commands

The library contains built-in modules to access such functionality as time (time), getting system information (sys), querying the operating system (os), and many more.

Using the standard library where possible makes code easier to maintain and port to other platforms.

Why import a module?

Module types

Modules can be:

1. Created by you. You might need Python to complete a specific set of tasks or functions that are grouped together. If so, it can be easier to bind them together into a module.
2. External sources (created by others).
3. Prepackaged with Python and part of the standard library.
   - Examples: **math**, **time**, and **random**.

Importing modules

Standard library modules are imported by using the `import` command. You can also import specific functions or constants from a module by using the `from` command.

Examples include:

```
import math
-> use math.pi to access the pi constants
-> use math.exp(x) to access the exp function
from math import pi
-> use pi directly in your code
from math import exp
-> use exp directly in your code
```

You must import the module or the function before you can use it, even if the module is part of the standard library.

Creating modules

To make your own module :

- Create a file name with a .py extension (for instance mymodule.py)
- Add your code to define some functions
- You can now import mymodule in other python files and use the code defined in the module

mymodule.py

```
1  import math
2
3
4  def calculate_circle_area(radius):
5      return math.pi*radius**2
```

testmodule.py

```
1  from mymodule import calculate_circle_area
2
3  print(calculate_circle_area(5))
```

Run

```
78.53981633974483
```

Demonstration: Math Module

10

Instructions

1. Create a file that is called: **modules.py**
2. Import the **math** module.
3. Try the absolute value function: `math.fabs(x)`
4. Next, try the floor function: `math.floor(x)`
   - This function takes a float and returns the largest integer that is less than or equal to *x*.
5. Print your results and run the file.

Extra challenge:

1. Can you customize your **print** statement to describe what you did?
2. What happens when you try to put a string into `math.floor(x)`?

aws re/start

The following sample code can be used for this demonstration:

```
import math

absolute = -5.999
floor_test = 198.42

result1 = math.fabs(absolute)
result2 = math.floor(floor_test)

print(result1, " is the absolute value of ", absolute)
print(result2, " is the flow of ", floor_test)
```

File handlers

Use the built-in **`open()`** function.

Two arguments, both strings—the path to the file, and the file mode (read or write)—return a file handler object that represents that file.

File handlers can read data from the file or write data to the file. They can also pass data to other functions that can do so on your behalf (like **json.load** and **json.dump**).

File handlers must be closed after use, by calling the attached **`.close()`** function.

This example shows how file handlers are used to open, read, write, and close files. Line by line, the code proceeds as follows:

1. A file handler that is named *f1* is created by calling *open* on the file *diary.txt* in read mode (which is designated by the *r* argument).
2. The file handler reads the contents of the file. In this case, the file contents are written to the console.
3. Contents of the *diary.txt* file appear in the console.
4. Close the handler to the file. This step is important because, while the handler is open, it is using system resources. Closing the handler releases the resources back to the operating system for
5. *f2* is used to open the file *diary2.txt* in write mode (which is designated by the *w* argument).
6. A string is written to the file by calling **`write()`** on the f2 file handler.
7. Close the handler. Again, this step is important. If the handler remains open it will reduce system performance by not releasing system resources back to the operating system for other processes.

Exception handling

- Exception handling is another form of flow control.
- When an error occurs, instead of failing and quitting the program, you can use a **try/except** block.
- You must specify the exception (or exceptions) that you expect might occur.

aws re/start

Examples: Exception handling

OS: Operating system module

- OS is part of the Python standard library.
- The OS module provides operating system functionality. The output of the module depends on the underlying operating system, with generally the same inputs.
- Common capabilities in the OS module are environment variable information, file manipulation, directory traversal, and process management.
- Programs that import and use the OS module are generally more portable between different platforms.

OS capabilities

- Host operating system:
  - **getlogin** – Returns the name of the logged in user
  - **getgrouplist** – Returns a list of group IDs that a user belongs to
  - **getenv** – Returns the value of the environment variable that is passed to it
  - **uname** – Returns information to identify the current OS
  - **system** – Is used to run commands in a subshell of the system
- Common functions for files:
  - **chown** – Changes the ownership of a file
  - **chmod** – Changes the access permissions of a file
  - **remove** – Removes the file at the given path
- Common functions in the os module for directories:
  - **getcwd** – Gets the current working directory
  - **listdir** – Lists the contents of the current directory
  - **mkdir** – Creates a new directory

OS: Run commands on the host system

```
>>> os.system("command on the system to run“)

>>> os.system("adduser newuser")

>>> os.system("whoami")

>>> os.system("powershell.exe")
```

- Format that is used
- Linux command to add a user
- Linux or Microsoft Windows: Displays the current user
- Generally only for Microsoft Windows

Example: OS module – directory

```
>>> import os
>>> os.getcwd()'/home/username/folder'
>>> os.listdir()'["oldfolder"]'
>>> os.mkdir("newfolder")
>>> os.listdir()'["newfolder", "oldfolder"]'
```


The line-by-line explanation of the code explanation is as follows:

1. Import the os library.
2. To get the directory that the code runs in, use the **`getcwd()`** function from the *os* library. The output from the **`getcwd()`** call displays the current working directory of '/home/username/folder'.
3. Calling **`listdir()`** lists the contents of the current directory. The output from the **`listdir()`** call displays the contents of the current directory. In this case it is 'oldfolder'.
4. Calling **`mkdir()`** creates a new directory. In this case, it creates a directory that is called *newfolder*.
5. Again, **`listdir()`** is called to list the contents of the directory. The output from **`listdir()`** shows the *oldfolder* entry and the new *newfolder* entry.

Demonstration: Work with the OS Module

19

Instructions

1. List your current working directory.
2. Create and then delete a new folder in your current working directory.
3. Print information about the currently logged-in user.

aws re/start

1. From the command line, start a Python console by entering *python* and pressing ENTER.
2. Run the following commands:

```
import os
os.listdir()
os.mkdir("newFolder")
os.listdir()
os.rmdir("newFolder")
os.listdir()
os.getlogin()
```

JSON

- JSON stands for JavaScript Object Notation.
- JSON is a standard file format that transmits data objects. It is language independent. It was originally derived from JavaScript, but now most modern languages include the ability to generate and parse JSON (including Python).
- JSON is used to *serialize* objects, which means that it turns code into a string that can be transmitted over a channel.

```
{
 "users": [
  {
  "name": "John Doe",
  "age" : 25
  },
  {
  "name" : "Li Juan",
  "age" : 29
  },
  {
  "name" : "Sofía Martínez",
  "age" : 22
  }
 ],
 "dataTitle": "JSON Tutorial!",
 "swiftVersion" : 2.1
}
```

JSON, continued

- Four useful functions in the JSON module are:
  - **dump**
  - **dumps**
  - **load**
  - **loads**
- **dump** and **dumps**: Turn various kinds of structured data into a string, which can be written to a file.
- **load** and **loads**: Turn a string back into structured data.
- **dump** and **load** work directly with files.
- **dumps** and **loads** work with strings.
  - The *s* at the end of the name is for *string*.

Uses for JSON

As mentioned earlier, JSON is used to make large amounts of data into strings and then put them back into their correct data types.

JSON takes a float and turns it into a string so that the information can be sent easily. Then, it turns the string back into a float when the information is received.

This technique is especially useful when you have more than one data type in a dataset. You can translate your Boolean, Integer, Float, and String to a single string.

You can then store the information in a file or a webpage.

When the data is retrieved and turned back, they are still the Boolean, Integer, Float, and String data types.

Example: JSON module

```
>>> import json
>>> json.dumps(5)
'5'
>>> json.dumps([1, "string", 3])
'[1, "string", 3]'
>>> json.loads('[1, "string", 3]')
[1, "string", 3]
```


The line-by-line code explanation is as follows:

1. Import the JSON library.
2. The **`dumps()`** call converts the indicated value into JSON.
3. The output from the conversion of *5* to JSON displays. In this case, it is the number *5*.
4. The integers *1* and *3* are converted, along with the string *string*.
5. The output of the conversion displays.
6. The **`loads()`** function from the Python JSON library converts a JSON string into a Python dictionary.
7. The output of the **`loads()`** command displays.

Demonstration: Recognize and Greet People by Name

Instructions

1. Use the JSON module to save and recall the history of past runs.
   a. Create **helloworld.py**.
   b. Import the JSON module.
   c. Write data to a file.
   d. Examine the file.
2. Use the if statement for flow control.
   a. Update helloworld.py to read from the file.
   b. If the file has data, use it in the greeting. Otherwise, request the user's name.
3. Try/except blocks are used to handle exceptions.
   a. Update the program again to use a try/catch block to handle file errors

The following code covers all the requirements in the instructions:

```
import json

filename = 'userName.json'
name = ''

# Check for a history file
try:
    with open(filename, 'r') as r:
        # Load the user's name from the history file
        name = json.load(r)
except IOError:
    print("First-time login")

# If the user was found in the history file, welcome them back
if name != "":
    print("Welcome back, " + name + "!")
else:
    # If the history file doesn't exist, ask the user for their
name
    name = input("Hello! What's your name? ")
    print("Welcome, " + name + "!")
```

```
# Save the user's name to the history file
try:
    with open(filename, 'w') as f:
        json.dump(name, f)
except IOError:
    print("There was a problem writing to the history file.")
```

What is pip?

pip is the package manager for Python, and it is similar to apt in Linux.

- It is used to install third-party packages. A package holds one or more Python modules that you can use in your code.
- It is installed along with Python.

It is not called from within Python—pip is called from the command line, like Python itself is.

Demonstration: Install Requests with pip

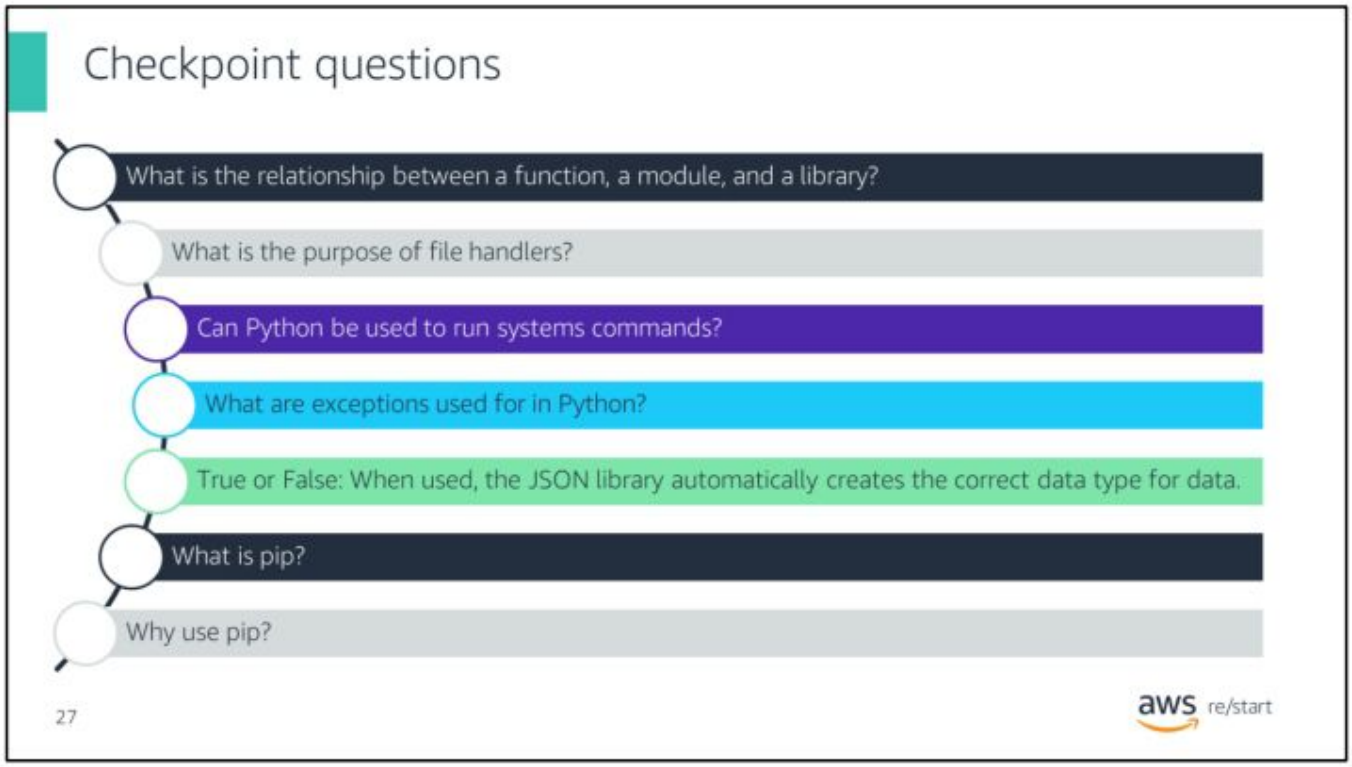
26

Instructions

1. Test to see whether *Requests* is already installed.
   - To test it, in the command line, enter: ***pip show requests***
2. Review the results.
3. Install requests with pip.
   - Enter the command: ***pip install requests***
4. Review the results.
5. Test whether Requests is installed.
   - Use the same command that you used previously.
6. Review the results.

aws re/start

- For steps 1 and 2, run *pip show requests* and review the output.
- For steps 3 and 4, run *pip install requests* and review the output.
- Rerun *pip show requests* and review the output.



Answers:

1. Functions are blocks of code that can be wrapped in a module with other functions. Multiple modules can then be wrapped into a single library.
2. File handlers enable Python to read and write to files.
3. Yes, by calling functions like `system()` from the os library.
4. When an error occurs in a Python application, an exception is raised. The exception can be caught and handled to enable the application to elegantly handle the occurrence of errors.
5. True.
6. pip is the standard package manager for Python. You can use it to install and manage additional packages that are not part of the Python standard library.
7. pip makes it easier to manage packages in Python than it would be to install, update, and remove them manually.

Key takeaways

- Python uses modules and libraries to store code that is used often or that is distributed to many systems.
- Modules like *os* enable developers to run operating system commands.
- The module *JSON* can create, read, and write to JSON files.
- pip can be used to quickly and easily manage Linux packages that Python uses.

aws re/start

Some key takeaways from this lesson include:

- Python uses modules and libraries to store code that is used often or is distributed to many systems.
- Modules like *os* enable developers to run operating system commands.
- The *JSON* module can create, read, and write to JSON files.
- Pip can be used to quickly and easily manage Linux packages that Python uses.

aws re/start

Python for System Administration

Python Fundamentals

Name of presenter

Date

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

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

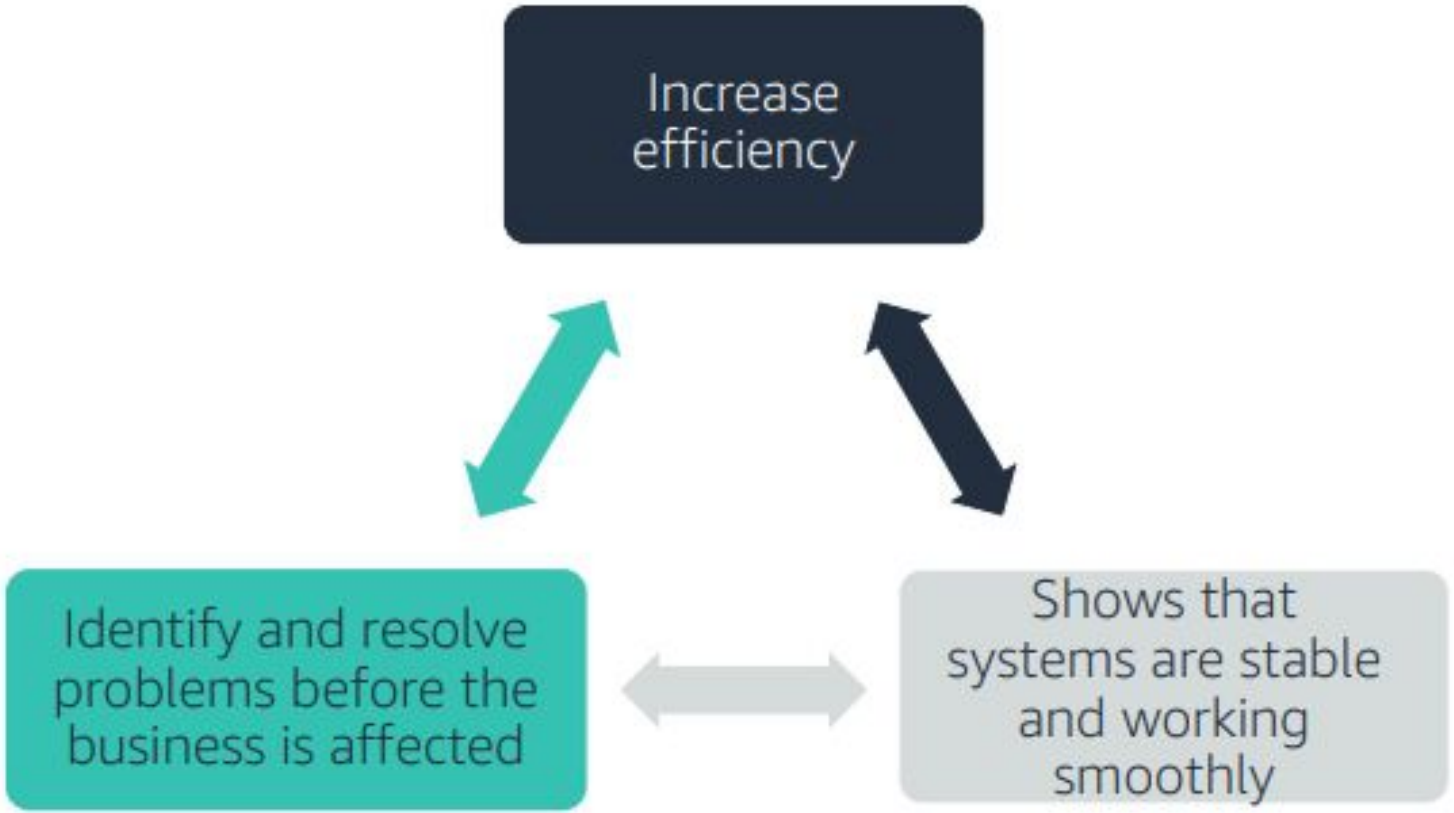
Includes these common tasks:

- 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



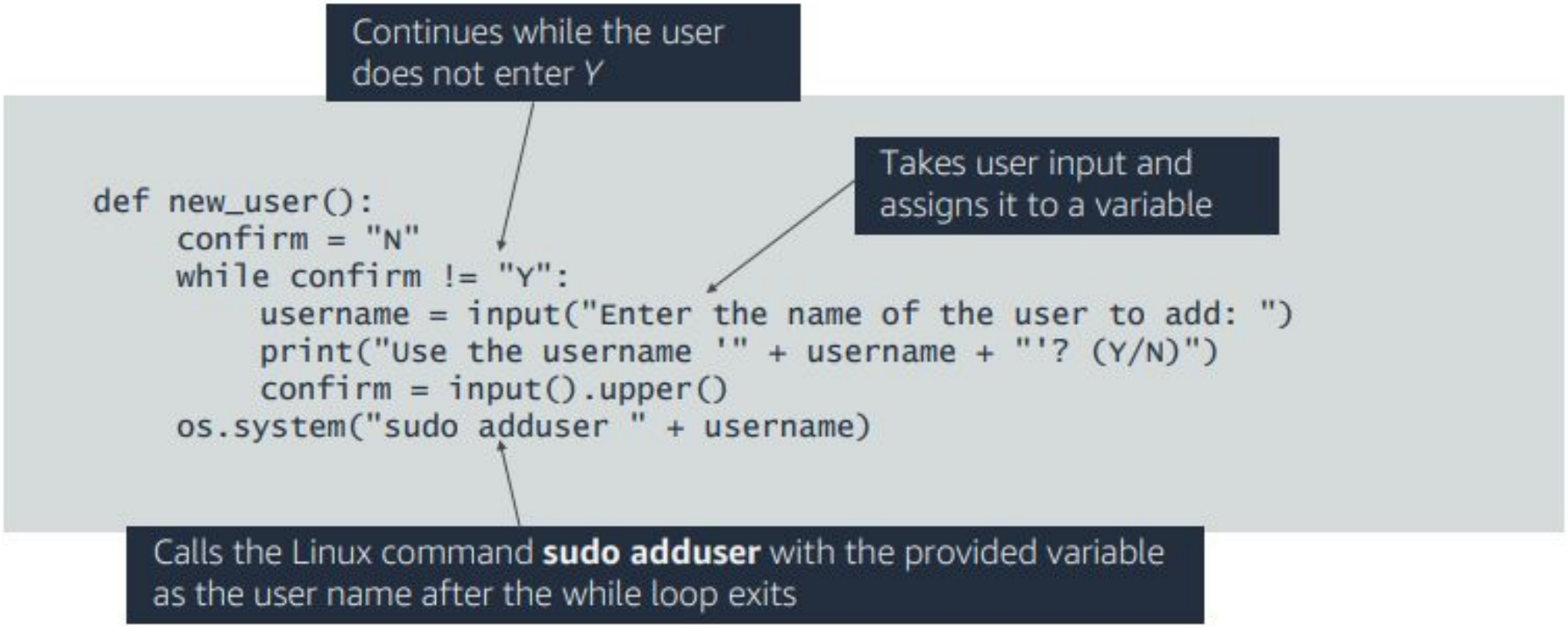
Activity: Working with Users

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



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 = str(input("Groups: "))
```

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 = str(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 = str(input("Groups: "))
```

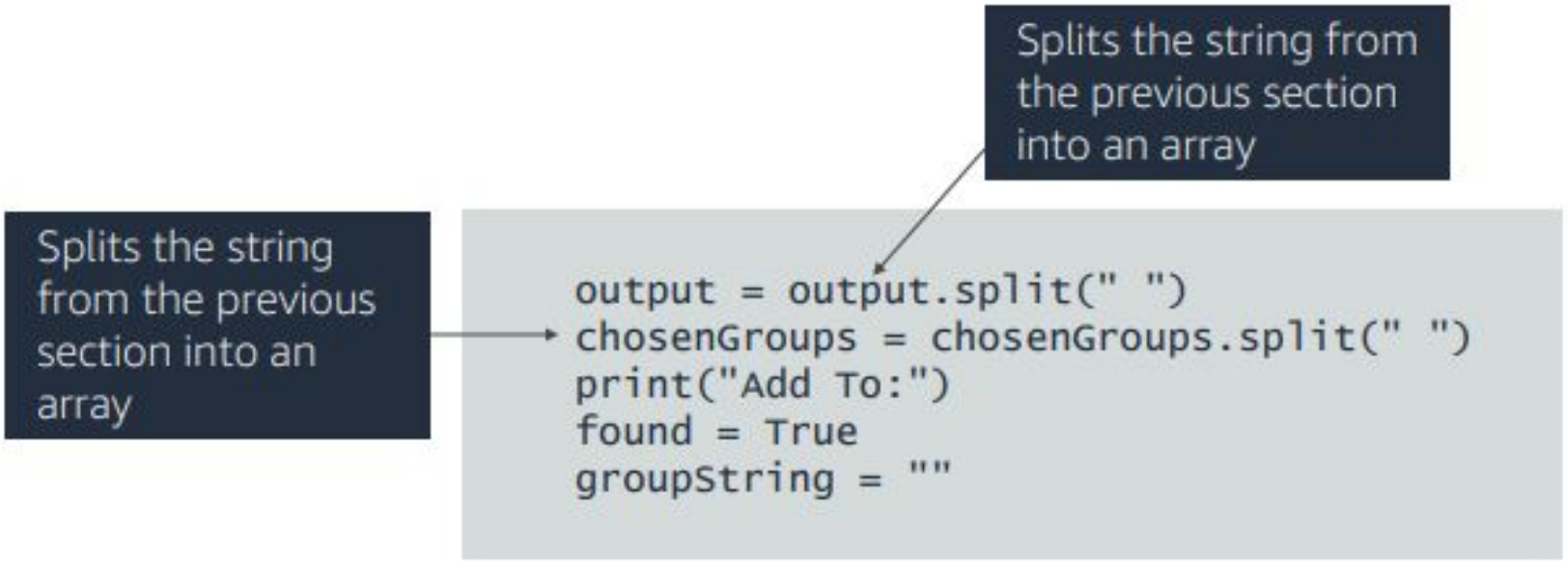
Takes the list of groups that the user should be added to

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

Activity: Adding a user to a group (2)

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

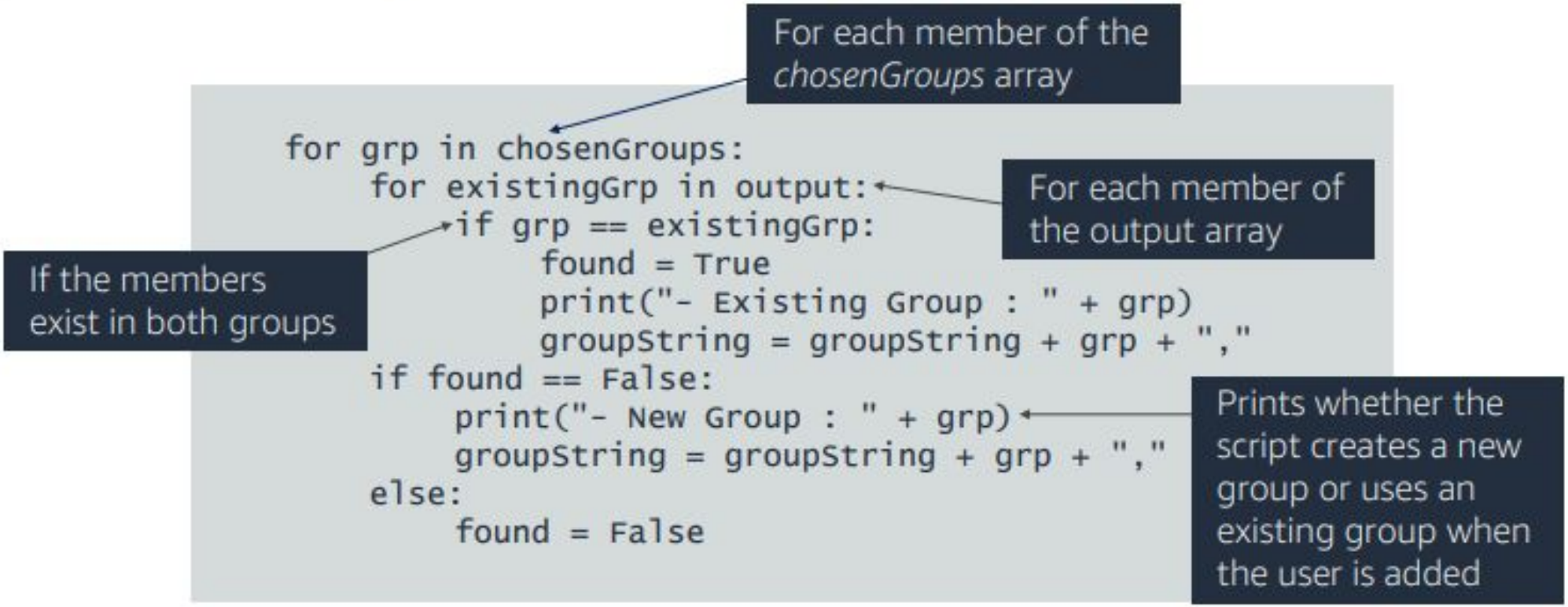
Activity: Adding a user to a group (2) – solution



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
```

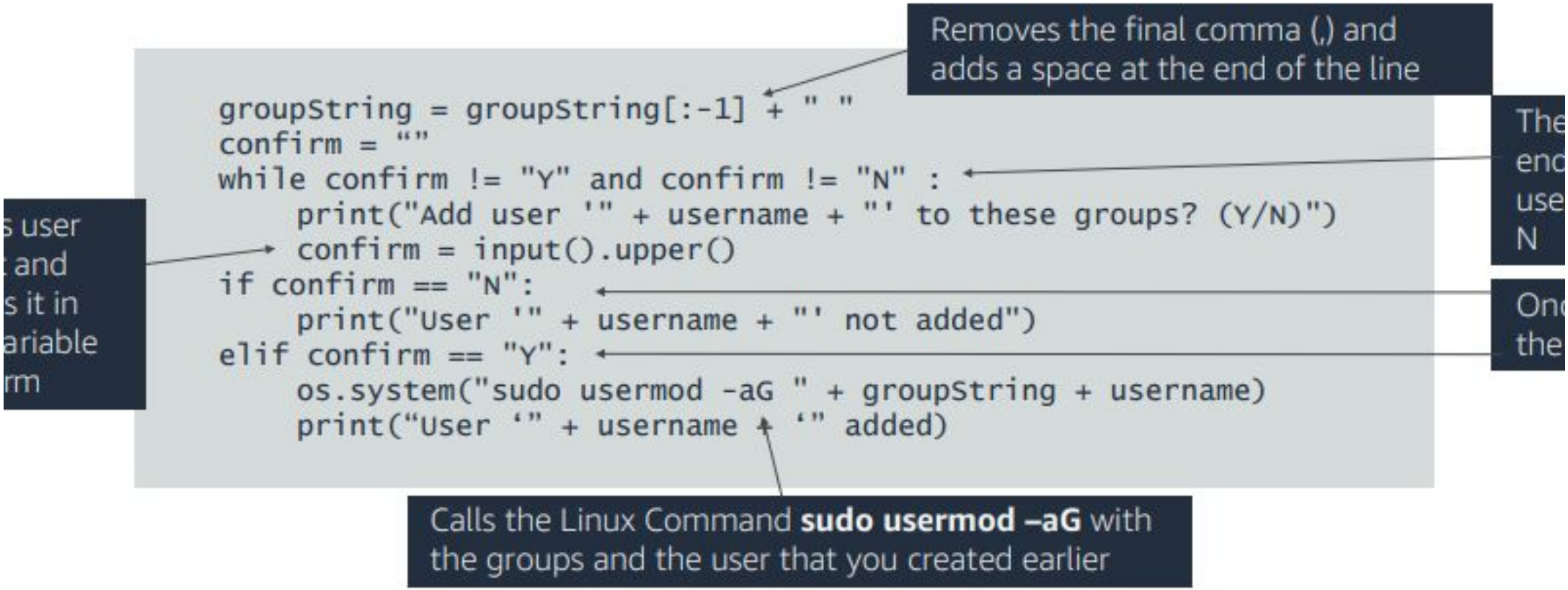
Activity: Adding a user to a group (3) – solution



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



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

```
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)
```

Describes how the input should be formatted

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)

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")
```

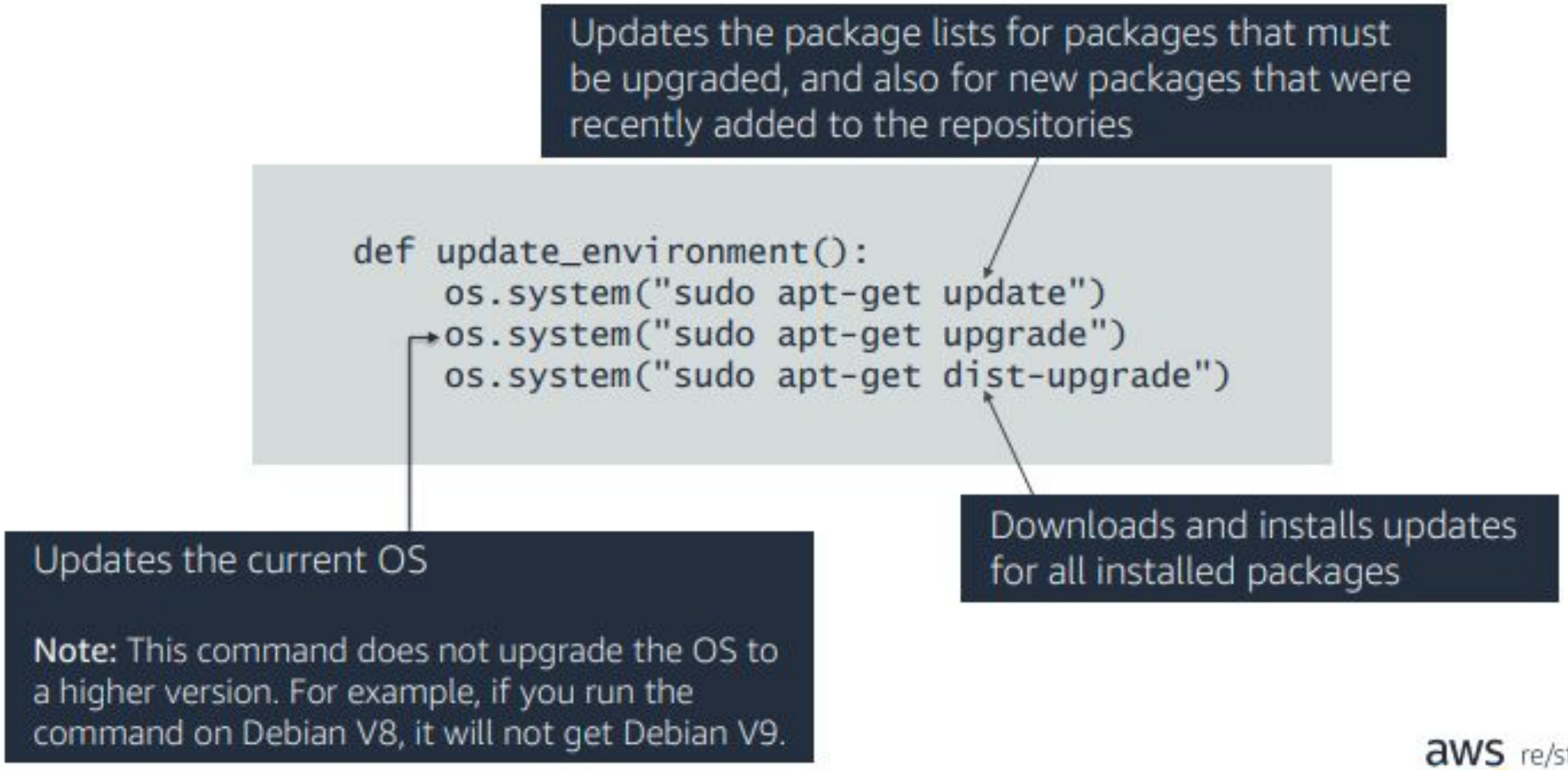
Cleans obsolete deb-packages

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")
```

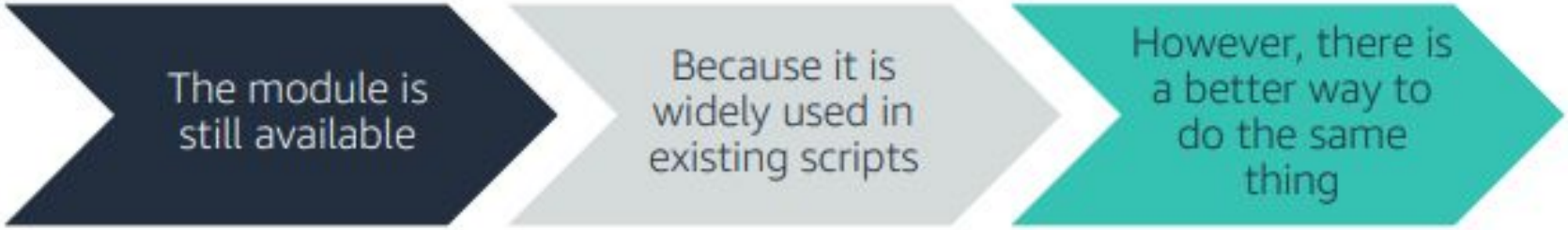
Activity: Handling packages (5) – solution



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

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

Module deprecation:



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()`

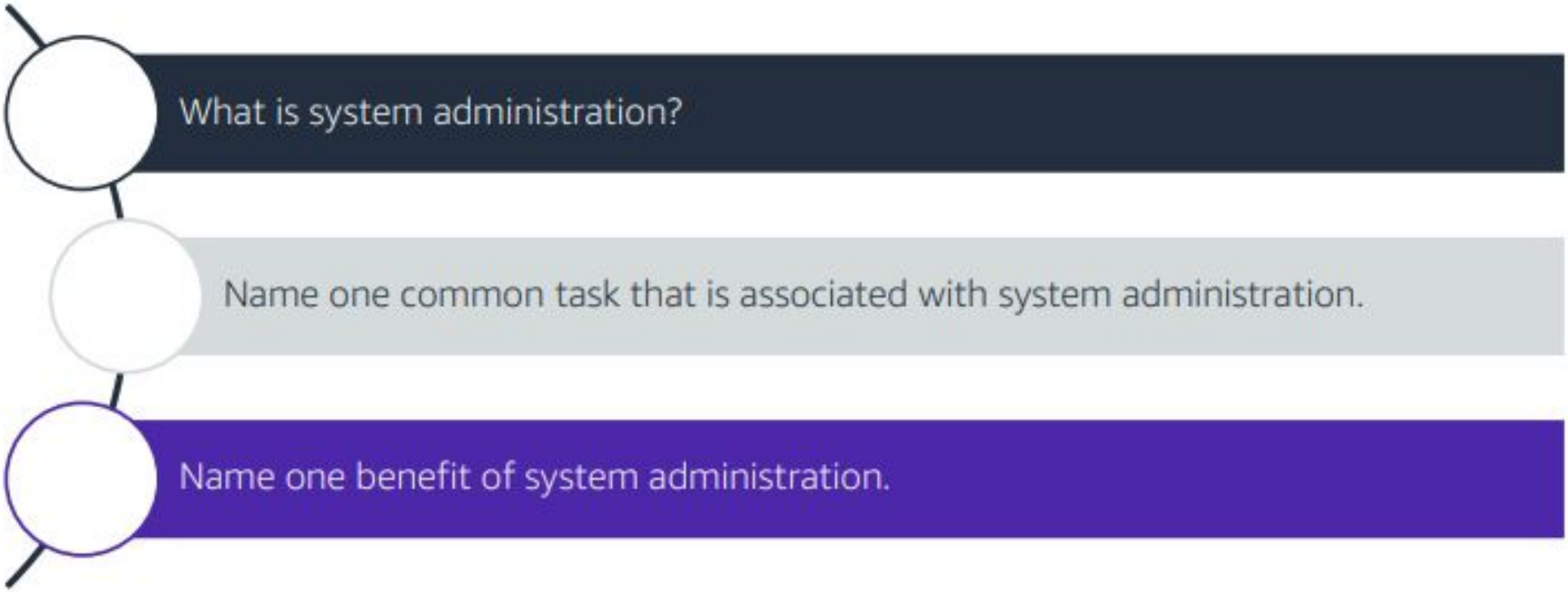
- By default, it does not use a shell. Instead, it tries to run a program with the given string as a name.
- You must pass in a list to run a command with arguments.
  - Example: `subprocess.run(["python","-version"])`

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



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.

Welcome to Debugging and Testing.

Debugging

What you will learn

At the core of the lesson

You will learn how to:

- Explain the purpose of debugging and testing code
- Explain how debuggers enable developers to find bugs in their code
- Recognize how to perform a static analysis on Python code
- Recognize how to evaluate Python code for ways to implement dynamic analysis

aws re/start

In this lesson, you will learn how to:

- Explain the purpose of debugging and testing code
- Explain how debuggers enable developers to find bugs in their code
- Recognize how to perform a static analysis on Python code
- Recognize how to evaluate Python code for ways to implement dynamic analysis

What is debugging?

Logging can be a useful tool for finding logical problems in code.

Debugging is done to identify defects in the code itself. It is simple in concept and it has two parts:

- Finding errors in your code
- Fixing those errors

Debugging can be complicated in practice. Every language has tools for debugging to mitigate this challenge.

PDB: The Python debugger

The Python debugger is activated by entering:

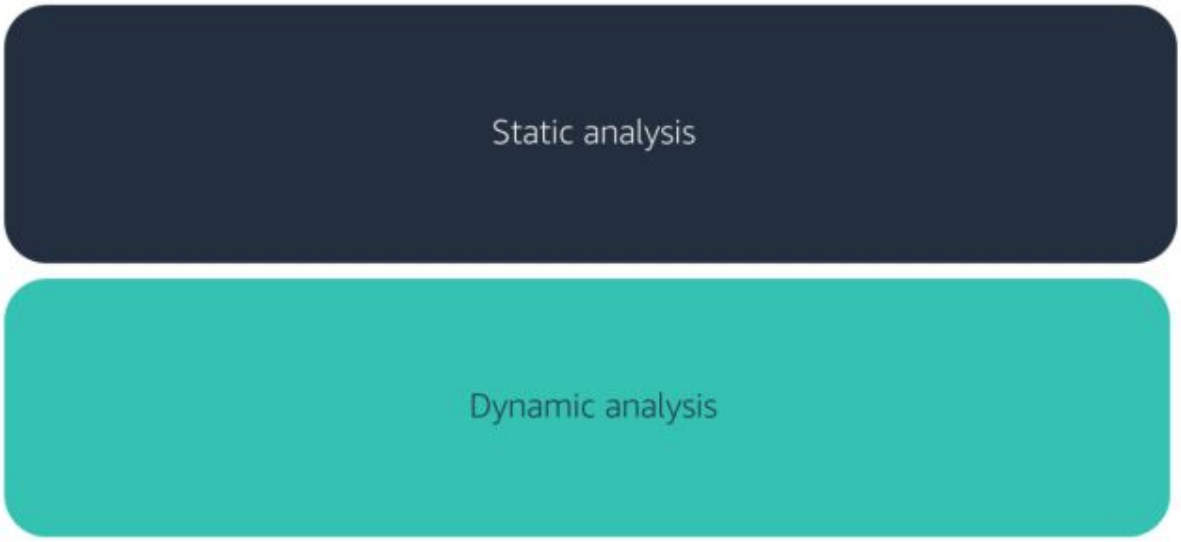
```
Python -m pdb <filename>
```

The first line of the code runs, and then a prompt (**pdb**) appears.

You can now use several commands.

All of these tools are used to find errors and help developers identify the reason behind them.

Types of debugging

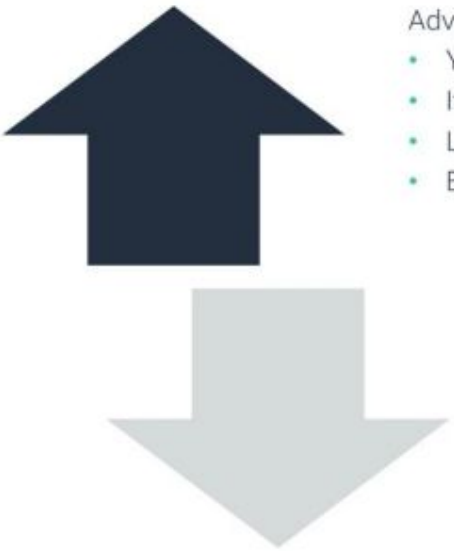


Static analysis

Static analysis has these characteristics:

- It can be done continuously in the development process.
- The Python interpreter includes a level of static analysis because it reports syntax and semantic errors. Integrated development environments (IDEs) can help identify issues while you write the code.
- It might consider factors such as proper nesting, function calls, and code complexity

Static analysis: Advantages and disadvantages



Advantages of static analysis include:

- You can identify the exact location of code issues.
- It has a faster turnaround time for fixes.
- Later tests have fewer issues.
- Earlier detection of bugs reduces costs.

Disadvantages of static analysis include:

- Manual analysis is time consuming.
- Automation tools can produce false positives and false negatives.
- Automation might result in taking security for granted.
- Automation is only as good as the parameters that are used to set up the tool.

Demonstration: Static Debugging

Instructions

1. Create a file that is called: **debug.py**
2. In that file, write a basic Python program based on a topic that you have learned. Make sure that you include a programming error. It could be a syntax error, a spelling error, or a different kind of error.
3. Run the program and show how the error is displayed.

aws re/start

Code example: The following code has two errors.

```
# Python program with 2 errors
var = "Double Value"
sumvalue = var + 4

def dosomething(valuetocheck):
    if valuetocheck > 4:
    print("Bad indent")
```

Dynamic analysis

Dynamic analysis has these characteristics:

- It is done by analyzing running applications
- Most IDEs for Python include a "debugging" mode. In this mode the application runs and a developer can execute code "line by line", until a particular line of code is called, or until a variable has a value (equal, less than, greater than).
- A invaluable technique during dynamic analysis is to write out values and conditions happening in a running application of a log file.

Dynamic analysis: Advantages and disadvantages

Advantages of dynamic analysis include:

- It identifies issues in a runtime environment (it could be a test server or a live version of the software).
- You can analyze applications even when you cannot observe the actual code.
- It can prove or confirm the false negatives that you identified from static analysis.
- It can be used with every application.

Disadvantages of dynamic analysis include:

- Automation might result in taking security for granted.
- You cannot guarantee full coverage.
- It can be more difficult to isolate code issues.

Assertions

- Assertions are conditions, such as *if* statements, that check the values in the application
- Dynamic analysis uses assertion statements during runtime to raise errors when conditions certain conditions occur.
- As an example consider the following function. The developer wants to ensure that the age value is always a positive number greater than zero. The following assertion checks this:

```
def loguserage(age):
    assert age <= 0, "Invalid age was supplied"
```

Log monitoring

Developers, in code, writes to a text file often referred to as a log file. As certain conditions happen in the running application, the logging code writes information to the log file

What does log monitoring do?

With log monitoring you get a full view of the application as it is running. The application can be exercised by a user and the developer can see inspect the log file for a "real world" use of the application.

What to log?

Consider every significant event in your application:

- Where did it occur?
- What time did it happen?
- What were the arguments?
- What is the state of important resources?

Capture all information when an error occurs:

- All arguments
- Exception, plus inner exceptions
- Traceback object: Stack traces

Log monitoring tools

In Python, the default, native log monitoring tool is the library *logging*.

To have access to the library

Enter *import logging* at the top of a Python file.

- That command enables access to customizable error messages that can be assigned different priority levels and then saved into a file for later review.

To save the output into a file

Add a line of code before you use the *logging* library:

```
logging.basicConfig(filename="app.log",
level=logging.DEBUG)
```

Demonstration: Dynamic Debugging

16

Instructions

1. Open the **debug.py** file.
2. Modify this file to take user input.
3. How can you dynamically test whether the value that the user supplies is greater than 0?

aws re/start

Code example:

```
# Ask the user for a value and confirm the supplied value is
greater than 0

def checkvalue(valuetocheck):
    assert (type(valuetocheck) is int), “You must enter a
number.”
    assert (valuetocheck > 0), "Value entered must be greater
than 0"
    if valuetocheck > 4:
        print("Value is greater than 4")


var = int(input("Enter a number greater than 0: "))
checkvalue(var)
```

Software testing

What you will learn

At the core of the lesson

You will learn how to:

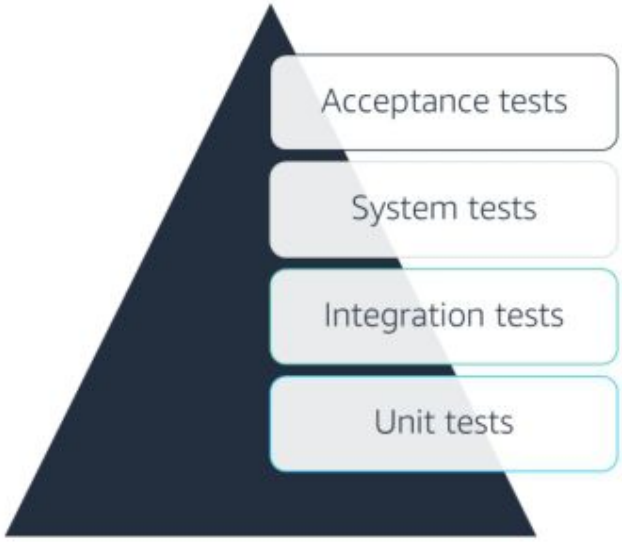
- Explain why software testing is necessary

aws re/start

In this lesson, you will learn how to:

- Explain why software testing is necessary

Testing


Acceptance tests
System tests
Integration tests
Unit tests

Unit tests

A *unit* is the smallest testable part of any software. It is the most basic level of testing. A unit usually has only one or a few inputs with a single output.

An example of a unit test is verifying each individual function of a program. Developers are responsible for their own unit testing.

Integration tests

Individual units are combined and tested as a group. You test the interaction between the different parts of the software so that you can identify issues.

Analogy: When a pen is manufactured, all the pieces of the pen (units) are produced separately—the cap, the body, the ink cartridge, and so on. All components are tested individually (unit testing). When more than one unit is ready, you can test them to see how they interact with each other.

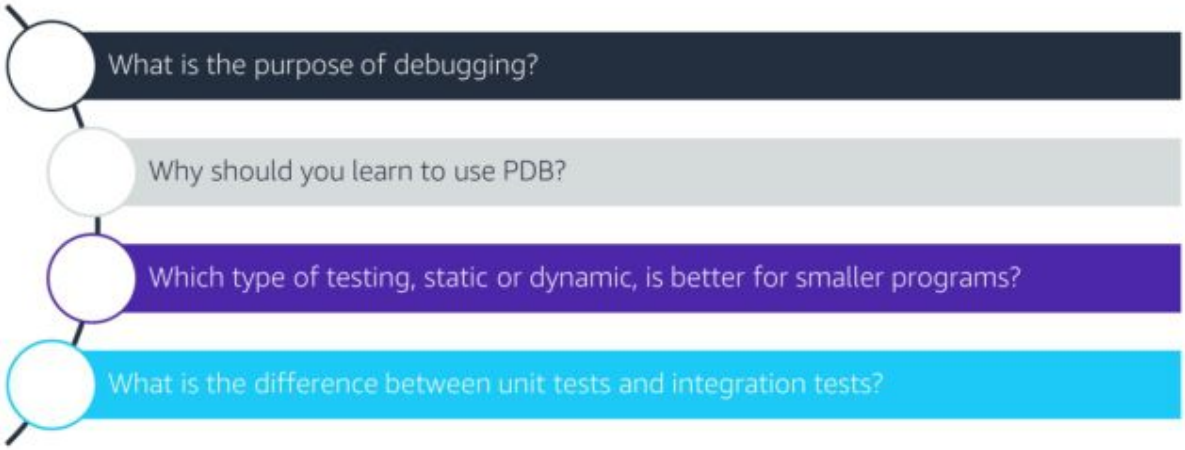
A developer can perform integration tests, but dedicated testers are frequently used to do these tests.

System tests

A complete and integrated application is tested. This level determines whether the software meets specific requirements.

Analogy, continued: When the pen is assembled, testing is performed to see if the pen works. Does it write in the correct color? Is it the specified size?

Checkpoint questions



aws re/start

Answers:

1. Debugging enables developers to verify an application's logic.
2. PDB, which is the Python debugger, enables developers to run various tools on their code for possible errors.
3. For smaller programs, static analysis is usually better because logic errors can be quickly identified in small code bases.
4. Unit tests cover code for functionality, and integration tests check the overall functionality of all parts of an application.

Acceptance testing

Acceptance testing is formalized testing that considers user needs, business needs, and whether the software is acceptable for delivery to the final user.

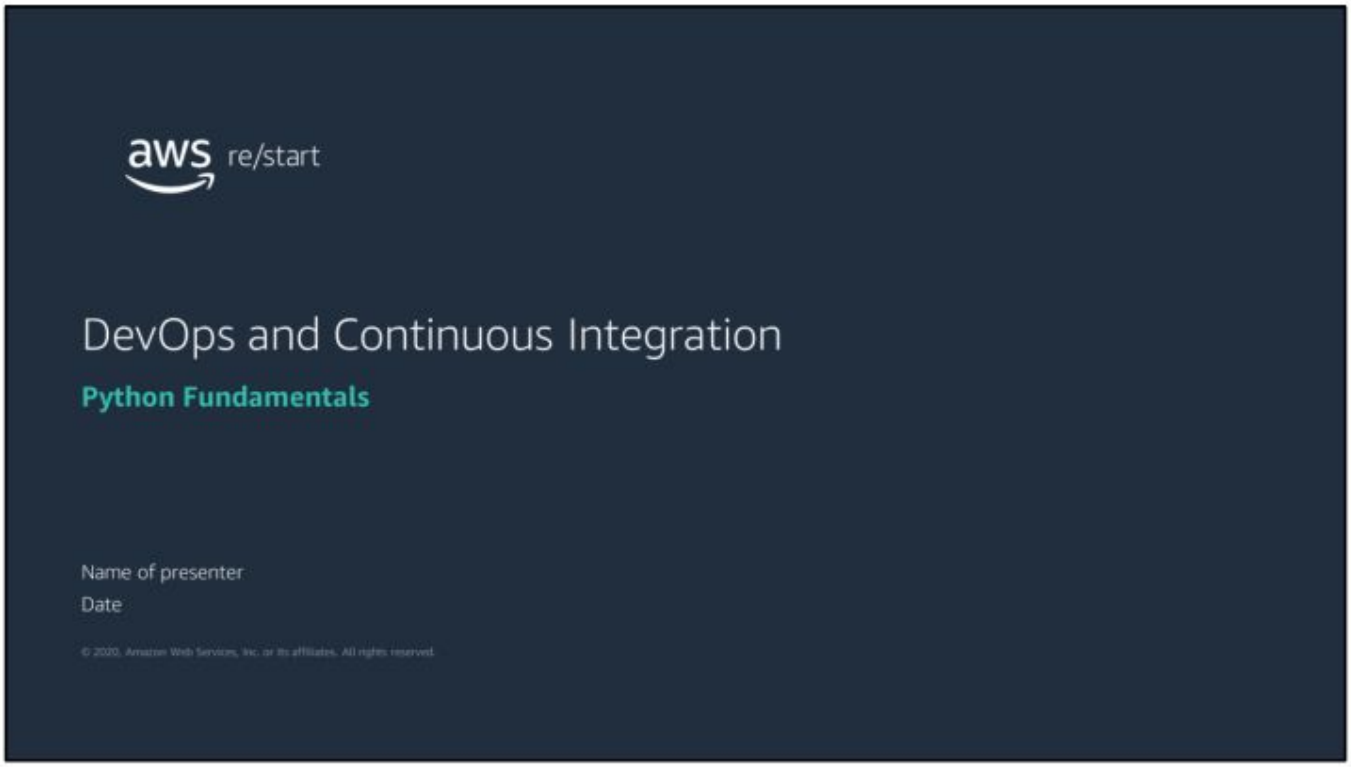
Key takeaways

- The continuous integration of code can include style checks and many forms of testing.
- Use logs to capture when errors occur in a program at runtime.
- Debugging is used to find errors in a program, and it can be done statically or dynamically.
- Many types of testing are done to ensure that applications work as expected, such as:
  - Unit testing
  - Integration testing
  - System testing
  - Acceptance testing

aws re/start

Some key takeaways from this lesson include:

- The continuous integration of code can include style checks and many forms of testing.
- Use logs to capture when errors occur in a program at runtime.
- Debugging is used to find errors in a program, and it can be done statically or dynamically.
- Many types of testing are done to ensure that applications work as expected, such as:
  - Unit testing
  - Integration testing
  - System testing
  - Acceptance testing



Welcome to DevOps and Continuous Integration.

Introduction to DevOps

What you will learn

At the core of the lesson

You will learn how to:

- Define DevOps
- Identify the goals of DevOps
- Identify the challenges that DevOps solves
- Describe the culture of DevOps

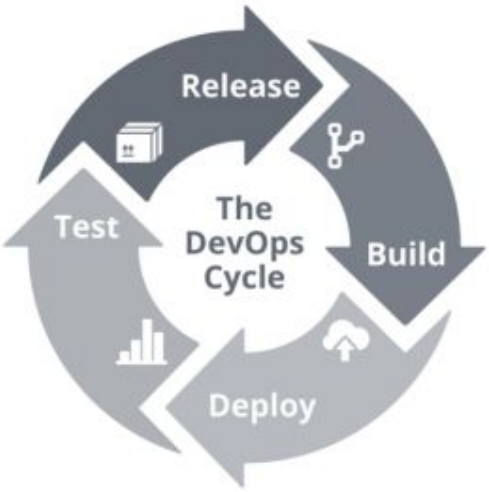
In this module, you will learn how to:

- Define DevOps
- Identify the goals of DevOps
- Identify the challenges that DevOps solves
- Describe the culture of DevOps

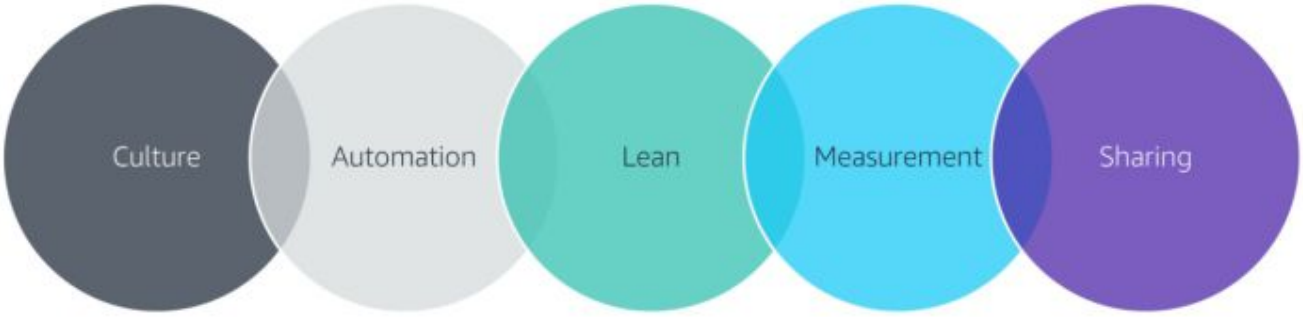
What is DevOps?

DevOps is a software engineering culture and practice that aims to unify software development (Dev) and software operation (Ops).

The main characteristic of the DevOps movement is to advocate for automation and monitoring at all steps of software construction. These steps range from integration, testing, and releasing to deployment and infrastructure management.



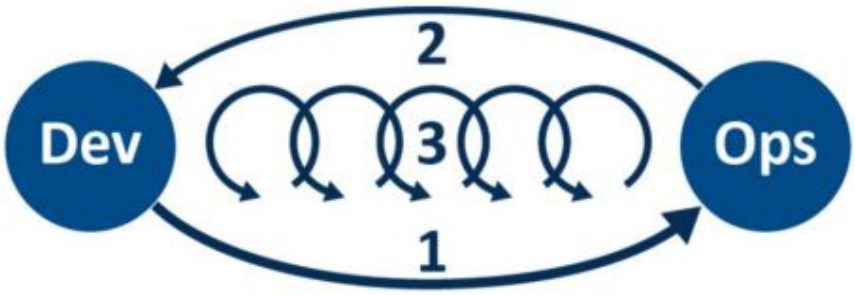
What is DevOps? continued



aws re/start

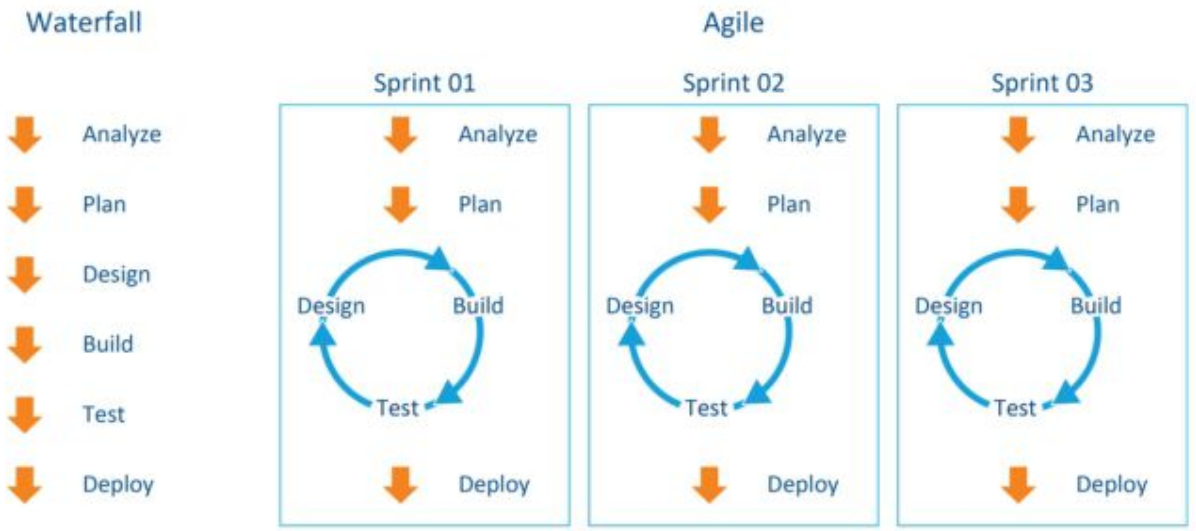
Culture of DevOps

1. Consider the performance of the **entire system – FLOW**
2. Create and amplify the right-to-left **feedback** loops
3. Create a culture that fosters:
    a. Continual experimentation, taking risks, and **learning**
    b. **Repetition and practice** – the prerequisite to proficiency



aws re/start

Waterfall versus agile



aws re/start

Goals of DevOps

- DevOps is meant to bridge the gaps between traditional IT, software development, and quality assurance (QA).
  - The most difficult part for beginners is the QA part. The appearance of your code is important.
- DevOps is meant to be faster and more flexible.
  - The challenge is better integration of QA and security in these quicker, shorter cycles (see the *Waterfall versus agile* diagram).
- DevOps is meant to bridge or reduce the need for specialized individual work.
  - When you begin to develop, you might notice that it is easy to become immersed in your own work. DevOps is meant to make it easier to do development work in a team.

Stakeholders: Organizational culture

Stakeholders: Organizational culture types

Collaborative culture revolves around collaboration.

Adhocratic culture is often associated with entrepreneurship and innovation.

Market culture pays attention to achieving goals, winning with the competition, and increasing measurable outcomes like market share or return on investment (ROI).

Hierarchical culture praises predictability, timeliness, and efficiency.

Continuous integration and continuous delivery (CI/CD)

What you will learn

At the core of the lesson

You will learn how to:

- Explain the need for automation
- Identify the states of CI/CD pipeline
- Describe continuous integration
- Describe continuous delivery
- Identify important features to look for in CI/CD tools

aws re/start

In this lesson, you will learn how to:

- Explain the need for automation
- Identify the states of CI/CD pipeline
- Describe continuous integration
- Describe continuous delivery
- Identify important features to look for in CI/CD tools

Automation

- When you develop software, it can be tedious and inefficient to perform the same tasks repeatedly. Automation can help solve this issue.
- The goal of automation is creative efficiency. However, automation has several risks that can undermine this goal:

- These concepts are explained in the next slide.

Automation: Risks

Over-automation

- Over-automation happens when you automate steps in the development process so that it reduces creativity. If you must think about and consider specific steps in a different way each time that you do them, you probably should not automate them—for example, analyzing, planning, and designing.

Under-automation

- Under-automation occurs when you avoid automation to make sure that things are handled correctly, or because it is helpful to find exactly where code stops working. Processes that are good to automate include building, testing, and deploying.

Bad automation

- Bad automation happens when you automate a process that does not work well. Bad automation can be fixed by revisiting the planning stage of development.

Tools for DevOps: Automation

Automation has many tools:

Automation encompasses many different methods and tools, but you are encouraged to focus on these three tools.

CI/CD pipeline

- The CI/CD pipeline is another tool for automation. The two parts are –
  - Continuous integration (CI)
  - Continuous delivery (CD)
- The next two slides cover CI and CD in detail.

Continuous integration (CI)

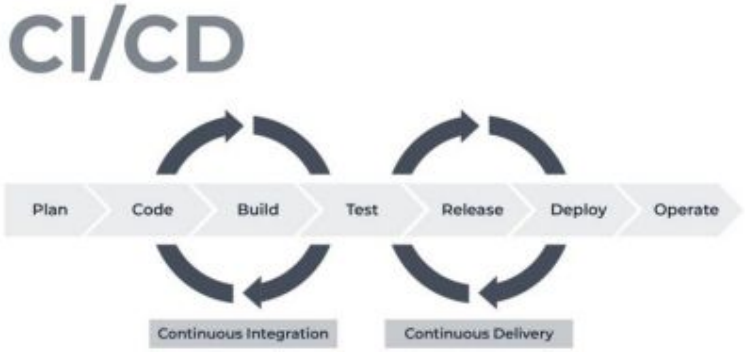
- CI is the automation of making your code available to your teammates.
- It generally includes the build automation and quality assurance automation that were discussed earlier.
- CI has two main purposes:

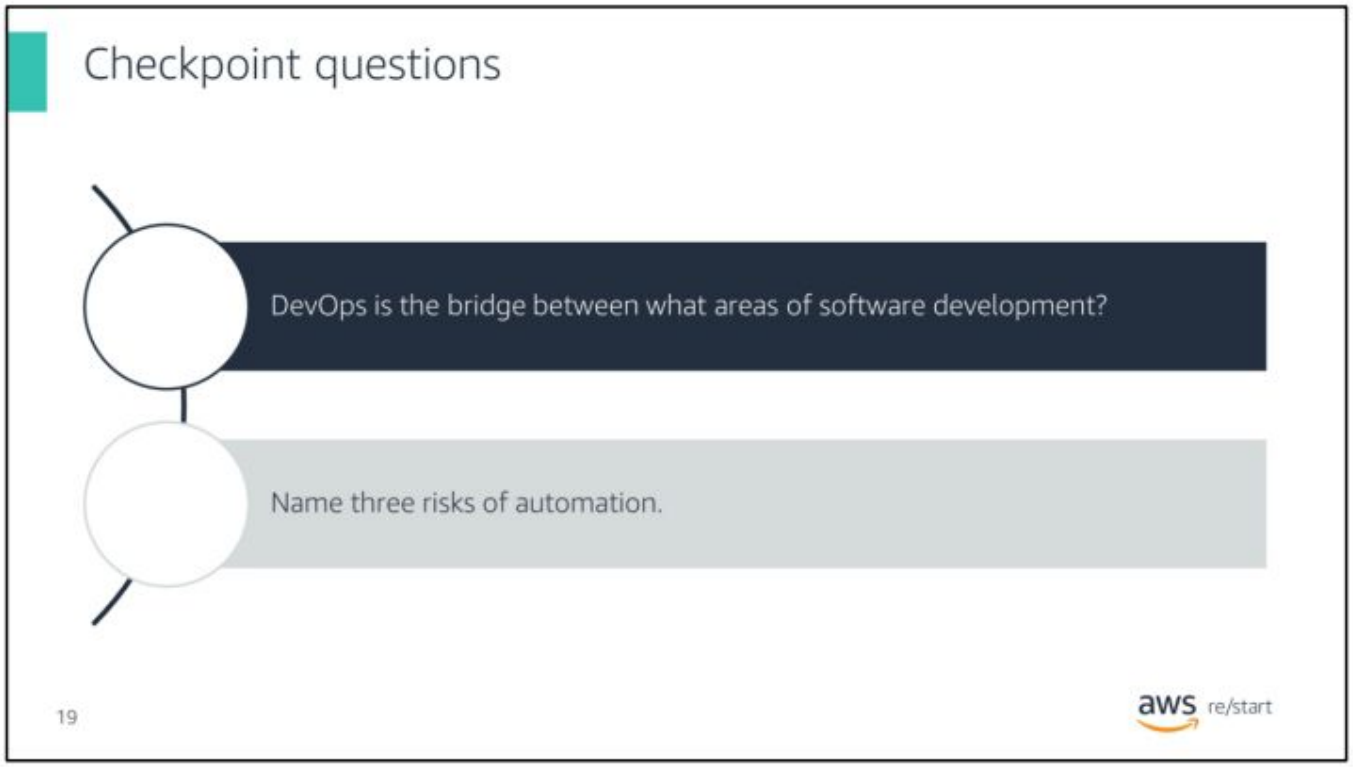
Making sure that your code works with what has already been done

Making sure that the code is readable for those who will work on it after you

Continuous delivery (CD)

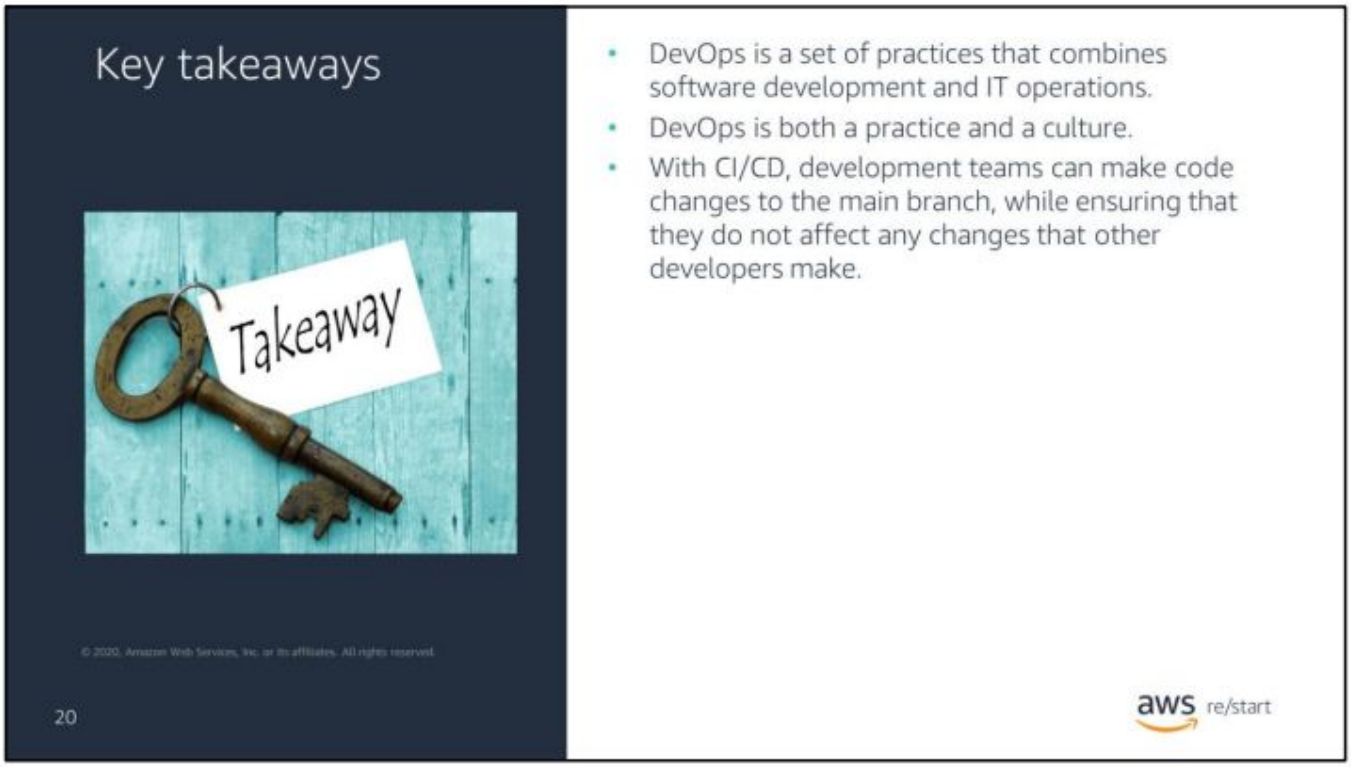
- CD is the extension of CI.
- CD includes a test automation for all code that is submitted. Its purpose is to ensure that the code works –
  - In the way that it was intended.
  - In a way that makes sense.
- CD also ensures that at any point in the development process, a working version of the code can be produced immediately.
  - This part is the deployment automation.





Answers:

1. DevOps bridges the gap between traditional IT, software development, and QA
2. Three risks of automation are:
    - Over-automation
    - Under-automation
    - Bad automation



Some key takeaways from this lesson include:

- DevOps is a set of practices that combines software development and IT operations.
- DevOps is both a practice and a culture.
- With CI/CD, development teams can make code changes to the main branch, while ensuring that they do not affect any changes that other developers make.

aws re/start

Configuration Management

Python Fundamentals

Name of presenter

Date

What you will learn

At the core of the lesson

You will learn how to:

- Define project infrastructure
- Explain why project infrastructure is important to the success of a project
- Define the purpose and function of software configuration management

aws re/start

In this module, you will learn how to:

- Define project infrastructure
- Explain why project infrastructure is important to the success of a project
- Define the purpose and function of software configuration management

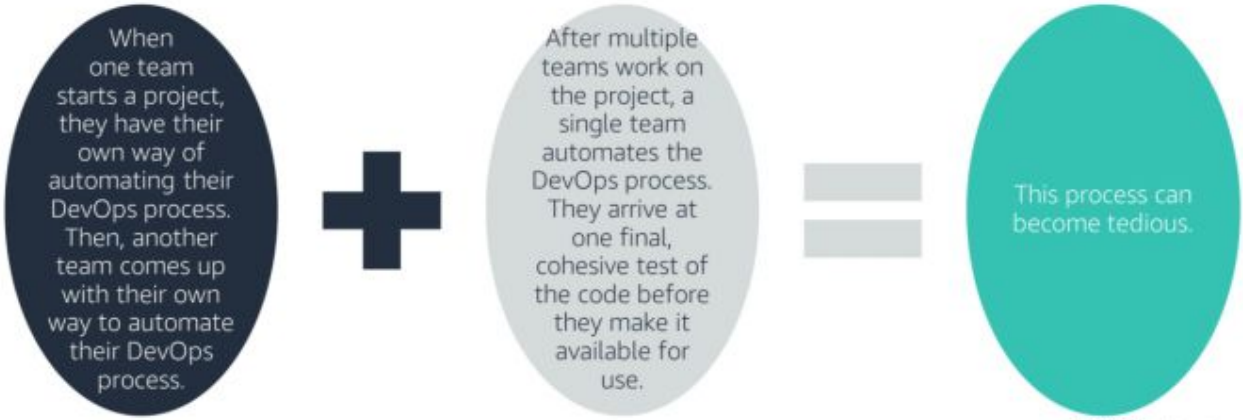
Project infrastructure

Project infrastructure

Project infrastructure is the way that a project is organized. An architect organizes the infrastructure of a bridge. Software developers organize the infrastructure of code.

Traditional project infrastructure

Traditional project infrastructure is a less-efficient way of developing code between teams.



aws re/start

Code organization

What does it mean to have code that is well organized?

Most companies have a certain coding style that their employees should follow. That style includes a way of naming variables in their code and the number of spaces to indent code blocks.

It is important to mention that each company varies in this practice. The details of the style matter less than following the style. It is more important for employees to follow the style so that they can avoid confusion.

There shouldn't be more than one set of tests for the various teams on a single project. That set of tests includes logic tests and compiling code.

Tools and templates

Many tools are available to build a cohesive infrastructure across teams.

For style

- Utilities like **pylint** can be run to ensure that code blocks are indented correctly, and to fix the code blocks that are not formatted well.

For logic

- Utilities like **pytest** can be used to run tests to make sure that code changes still meet the requirements.

Software configuration management

What is configuration management?

- Tracks versions of the code as it is developed
- Enables developers to work independently on different parts of a project, and then merge changes back into the project
- Version control software (such as Git) tracks what code changed and who made the changes
- When errors occur, configuration management enables fast rollbacks to previous, functioning versions

How does configuration management work?

- Developers *check out* code from a repository like AWS CodeCommit or GitHub.
- When they are finished with the code, developers upload their changes to the repository.
- When the new code passes all tests, it can be *merged* back into the main project.
- The process of checking code in and out can be done by:
  - Running **Git** from the command line
  - Using tools that are built into integrated development environments (IDEs), such as PyCharm
- Running tools—such as **pylint** and **pytest**—can also be a part of the check-in process.

Configuration management by example

The following simple example uses Git for configuration management because other steps might be needed, depending on how the repository (repo) is configured.

Get a copy of the remote repo:	`$git @<examplerepo.org>:<username>/<sourcecode>.git`
Commit changes locally:	`$git –commit –m "Message about the changes."`
Push change back to the repo:	`$git push`

Configuration management versioning

- As developers update the code, the release managers who are responsible for distributing new versions of software can monitor the changes.
- After all tests and functionality are verified, the release manager creates a new distribution of the software based on the contents of the repository.
- Release managers can now *version* the software, which helps manage customer issues when problems occur. (This point is where *rolling back to a previous version* becomes important.)
- In most cases, versioning takes the form of a numeric value (for example: *Version 3.2.1*).

Configuration management accounting

- Sometimes team leads and managers ask for the status of a project.
- With configuration management, the team can quickly report on the ongoing efforts by inspecting check-ins, check-outs, and other activities in the project's repository.

Configuration management security

- Because access to a repository must be granted, it stops unauthorized persons from gaining access to source code
- Because access is logged, it is possible to learn –
  - Who checked out and checked in code
  - When the check-ins and check-outs were done
  - What changes were committed

Key takeaways

- Project infrastructure is a critical discipline for helping to ensure that projects reach their goals.
- Project infrastructure also includes ensuring that Python code is styled properly and that it functions as expected.
- **pylint** checks the style of code, and **pytest** runs tests to ensure that the code works as expected.
- Software configuration management involves the use of code repositories to manage the code that is used in projects.
- To work with code repositories, tools are built into IDEs, and you could also use the command line tool **Git**.

aws re/start

Some key takeaways from this lesson include:

- Project infrastructure is a critical discipline for helping to ensure that projects reach their goals.
- Project infrastructure also includes ensuring that Python code is styled properly and that it functions as expected.
- **pylint** checks the style of code, and **pytest** runs tests to ensure that the code works as expected.
- Software configuration management involves the use of code repositories to manage the code that is used in projects.
- To work with code repositories, tools are built into IDEs, and you could also use the command line tool **Git**.