Methods in Open Science

Coding

Image from NASA: https://commons.wikimedia.org/wiki/File:Earth_Western_Hemisphere_transparent_background.png#filelinks

WV VIEW

AmericaView
Empowering Earth Observation Education
AmericaView.org - Est. 2003

This module explores key concepts associated with computer languages and coding or scripting. We will not investigate a specific language; instead, we will investigate general concepts. Later modules will provide specific examples using the Python and R languages, which are currently the most commonly used languages in data science applications.

Coding Languages

2

We will begin with a broad overview of how computers think and represent information and how coding languages are used to provide instructions.

Base-10 Number System

- Decimal system
- Only 10 symbols available: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Reuse symbols via positional notation
- Each new digit added to the left must have a value 10x greater than the digit to its right (e.g., 20 vs. 2)
- Probably developed because we have 10 fingers

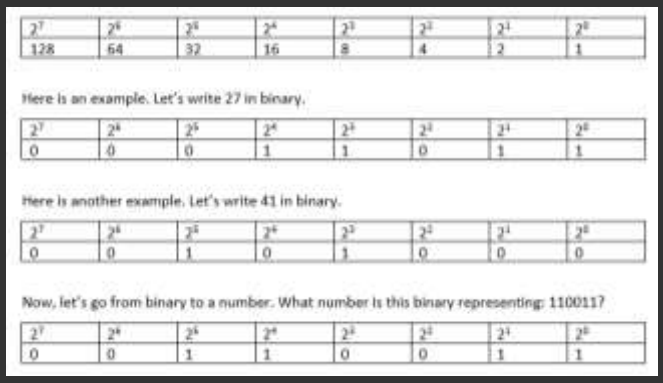
3

Humans commonly rely on a base-10 number system in which only ten different symbols are available (0 through 9) and these symbols are combined with positional notation to denote a value. Each place represents a magnitude of 10. Or, 10 is 10-time larger than 1 and 100 is 10-times larger than 10. So, understanding the meaning of the symbol and the position of the symbol in the sequence allows individuals to consistently represent numeric values.

Why do we use a base-10 system? This is most likely because we have 10 fingers on which to keep track of counts and quantities. If we did not have 10 fingers, we may have devised a different system. For example, if we had 12 fingers, we may have devised a system that used 12 unique symbols and positional notation.

Base-2 Number System

- Just two symbols: 0, 1
- Still use positional notation
- Each digit must have a value 2x greater than the digit to its right



2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

Here is an example. Let's write 27 in binary.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	1	1	0	1	1

Here is another example. Let's write 41 in binary.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	1	0	1	0	0	0

Now, let's go from binary to a number. What number is this binary representing: 1100112

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	1	1	0	0	1	1

4

Unfortunately, base-10 numbers are not optimal for a computer. In order to represent values or data within a computer, the information must be represented as some form of electrical signal, or a value must be represented electrically.

Computers make use of transistors, which have two states: On or Off. In other words, electricity passes through (On) or it does not (Off). By assigning the On state a value of 1 and the off state a value of 0, we have the beginnings of a language that computers can understand: the presence or absence of an electrical signal.

So, there are effectively two symbols or states that can be physically represented by a transistor: On or Off (i.e., 1 or 0).

In order to use these two conditions to represent numeric values, we cannot rely on a base-10 system, since only two states are available. Instead, we must use a base-2 system. In this system, we still use positional notation; however, each subsequent position will represent a value twice that of the prior position as opposed to 10-times. As described on this slide, this results in the following sequence from 2 to the 0^{th} power to 2 to the 8^{th} power: 1, 2, 4, 8, 16, 32, 64, 128, and 256.

This is the basic concept of binary, which makes use of a base-2 system and can be physically represented as on or off signals from transistors.

Binary

- Computers only understand binary (0 or 1) or (OFF or ON)
- Based on physical flow of electricity through a transistor:
  - 0 = no flow of electricity
  - 1 = flow of electricity
- Bit = 1 transistor (ON or OFF)
- Byte = 8 transistors

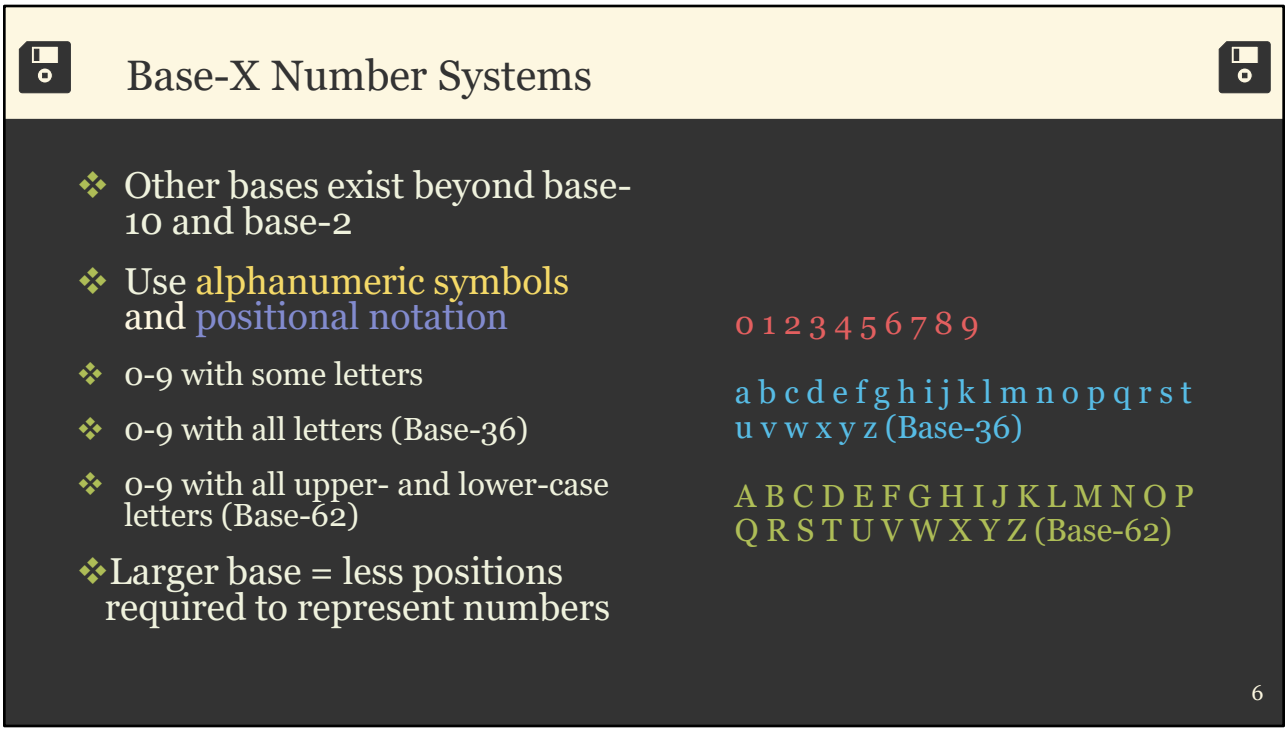
https://commons.wikimedia.org/wiki/File:Computer_Transistors_1965.jpg

5

Computers can only understand binary: 0 vs. 1 (or Off vs. On) (or, more specifically, voltage differences), as physically represented by the flow of or lack of flow of electricity through a transistor.

The signal from a single transistor is know as a bit of data, either 0 or 1 (Off or On).

When we combine the signal from 8 transistors, the resulting data unit is known as a byte. Each bit in the byte represents one location in positional notation from 2 to the power of zero to 2 to the power of 7. Using this system, values between 0 and 255 can be represented.



Before moving on, I wanted to note that base-10 and base-2 are not the only possible number systems. A variety of bases can be used. After progressing past base-10, a combination of alphanumeric symbols are used to represented each allowed symbol. For example, base-32 uses all numeric values (0-9) and all letters (a-z). Base-62 makes use of all numeric values (0-9), all lower-case letters (a-z), and all upper-case letters (A-Z).

When using a larger base, less positions in positional notation are needed to represent a number.

Hexadecimal

- Base-16 Number System
- 0 to 9 and A through F
- Can convert between decimal, binary, and hexadecimal systems

Integer	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

7

As a specific example, hexadecimal is a base-16 number system. In this system, values 0-9 are used along with letters A through F to represent the possible symbols.

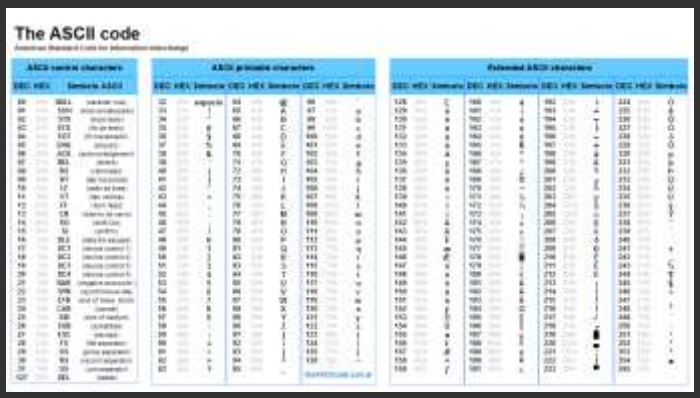
Remember that a base-2 system requires 8 positions to represent all values between 0 and 255. In contrast, hexadecimal can represent values between 0 and 255 with only two positions where each subsequent position is 16-times larger than the prior position.

For example, 0 is represented as 00, or (0x16) + 0 = 0. 255 is represented as FF, or (15 X 16) + 15 = 255.

Any value between 0 and 255 can be represented with just two positions. For example, 50 would be 32, or (3X16) +2= 50.

ASCII

- American Standard Code for Information Interchange
- Character assigned to a byte of binary
- 256 possible combinations (2^8)
- Represent up to 256 unique symbols



https://commons.wikimedia.org/wiki/File:Ascii-codes-table.png

8

So far, we have discussed how binary is used to represent numeric values using the base-2 number system and bytes, which can be represented by the on and off states of multiple transistors.

However, in a computer system, we would like to be able to represent not just numbers but a wide range of alphanumeric characters. The American Standard Code for Information Interchange (ASCII) system assigns the quantities 0 through 255 to different alphanumeric characters. Since the values 0 through 255 can be represented using a byte of data, this allows for up to 256 symbols to be represented with a byte of data, or on/off signals from eight transistors.

Unicode

- What if you need to be able to represent more characters (such as multiple languages)?
- Unicode = modern unified encoding system
  - Assigns a code to every character and symbol in every language
- Different encoding schemes (UTF-8, UTF-16, UTF-32, etc.)
- UTF-8 = variable length encoding scheme where each character is represented by a 1- to 4-byte code
- UTF-16 = fixed width encoding scheme in which each written symbol is represented by a two-byte code.
- characters → binary or binary → characters

9

ASCII is limited since only 255 characters or symbols can be represented. If we consider all the symbols and special characters used to represent all the alphanumeric symbols globally and employed in different languages, this is not enough.

Unicode is a modern means to encode symbols and characters that allow for a representation of a large number of symbols representing multiple languages.

Unicode uses different encoding schemes. In order for characters to be correctly rendered, the encoding scheme must be known. If the encoding scheme is known, it is possible to convert between binary and characters. Thus, computers are able to represent the symbols used in human languages and number systems using a language that they can understand: on/off signals from transistors.

Programming Languages

- Middleman between human language and machine language
- Allow you to communicate with the computer
- Many different options are available
- Example Languages: Bash, C, C++, C#, Fortran, IDL, Java, JavaScript, Julia, MATLAB, Perl, PHP, Python, R, Ruby, Rust, SQL

10

To summarize, computers understand binary information as represented using on/off signals from transistors. These physical signals are combined with the base-2 number system as a means for computers to represent numbers. Encoding systems are used to represent alphanumeric symbols used by humans as binary numbers. More complex types of data, such as audio files and images, can also be rendered to binary information using similar methods. Thus, computers can represent our data.

However, we still have an issue: computers use binary or machine language while humans do not. So, we cannot directly speak to a computer. For example, you cannot open a computer terminal and type the following message and expect the procedure to be performed:

“Computer, look at last months invoices and sum up all of the invoices that are still outstanding.”

Unfortunately, computers are not yet able to accept a wide variety of commands either written in a human language or verbalized. So, we must be able to meet our computers halfway using languages that both humans and computers can understand.

This is the purpose of programming languages, of which there are many with

different use cases and strengthens and weaknesses. Programming languages allow us to use a set of rules and alphanumeric characters to input interpretable commands or instructions to a computer.

Compiled vs. Interpreted

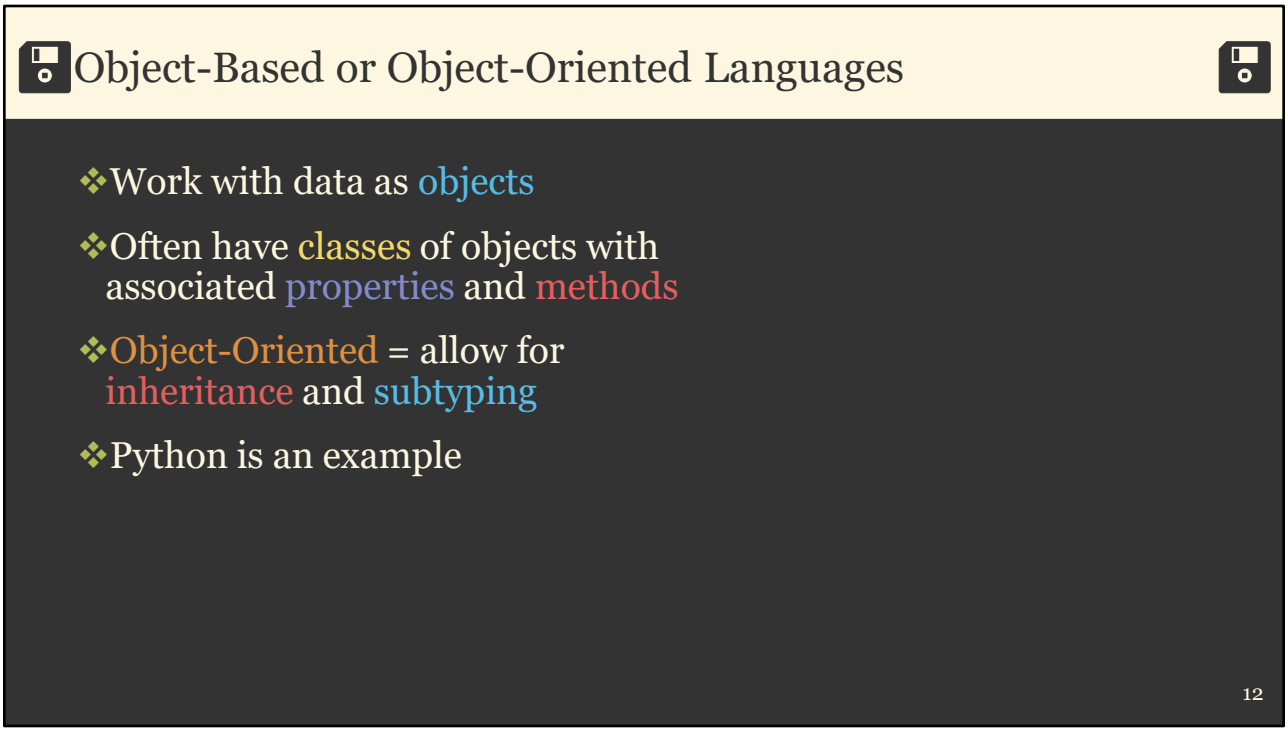
- Compiled = target machine translate the program/code directly
  - Language converted directly to machine code that process can execute
  - Fast and efficient
  - More control of memory management and CPU usage
  - Requires a build step to manually compile code
  - Platform dependent
  - C and C++ are examples
- Interpreted = intermediate program (i.e., interpreter) reads and executes program/code
  - Interpreter executes program line-by-line
  - Can be slower than compiled languages (using just-in-time compilation speeds up process)
  - Can be more flexible
  - Platform independent
  - Python and JavaScript are examples

https://www.freecodecamp.org/news/compiled-versus-interpreted-languages/ 11

Computer languages can either be complied or interpreted. A complied language requires the target machine to translate the program or code directly. In other words, the code is converted directly to machine language. Compiled languages tend to be fast and efficient and allow more control over the memory and CPU usage. However, they must be compiled before being used.

In contrast, interpreted languages make use of an interpreter, or intermediate program, to interpret the code line-by-line. This method can be slower than compiled languages, although methods such as just-in-time compilation speeds up the process. Such languages tend to be more flexible and do not require being compiled. Python is an example of an interpreted language.

Note that some languages can be implemented in both compiled and interpreted manners.



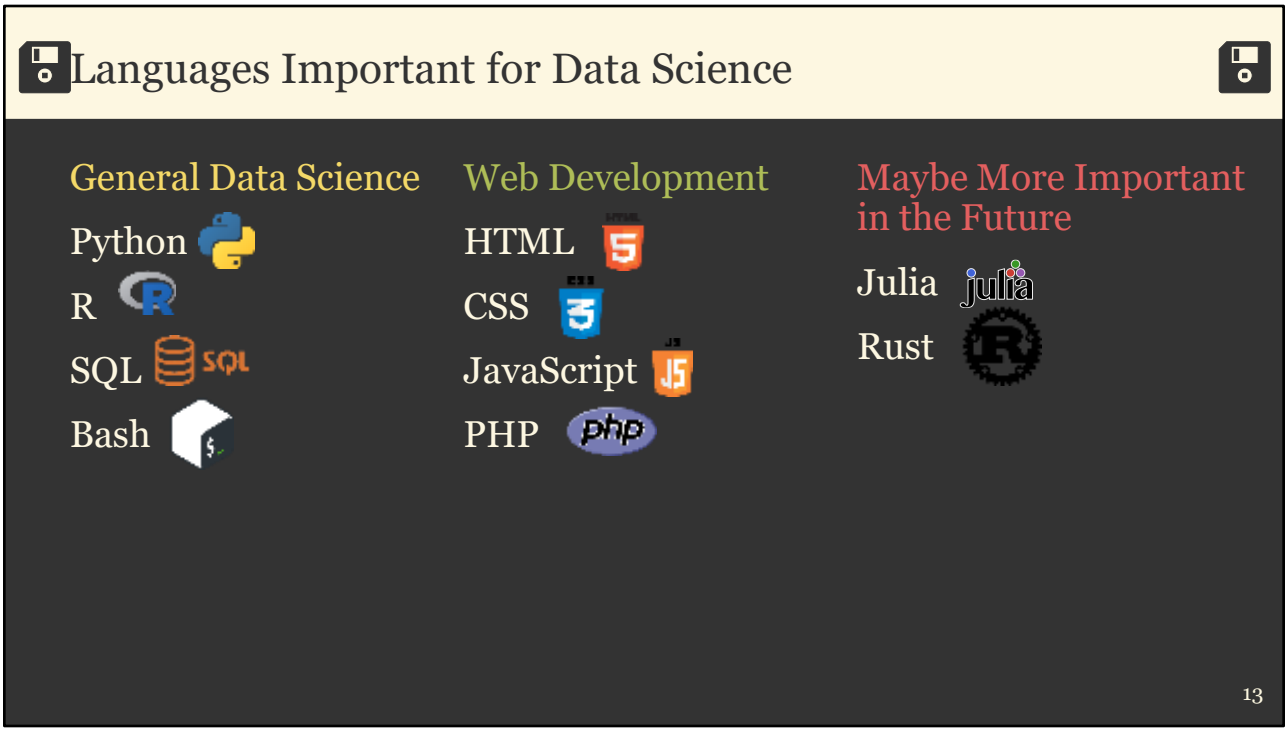
Object-based and object-oriented languages generally allow you to work with your data as objects. For example, an image could be represented as data stored in memory and referenced using a variable.

Different types of objects can be defined to represent different types of data, such as text, numbers, arrays, images, video files, 3D models, etc. These different types of objects are often termed classes, which have allowed characteristics, or properties, and methods that can act on them (i.e., functions associated with a specific instance of a class).

For example, a geospatial, or map, data layer could be represented as a class that has a property that represents the map projection and a method that allows for the user to obtain the spatial resolution of the data layer.

It is also generally possible to generate new classes that inherit characteristics from existing classes to act as more specific use cases. For example, a class representing geospatial vector data could be subclassed to create a more specific class that is used to represent only line vector features.

You will see object-based and object-oriented use cases as we work with Python in later modules.



As already mentioned, a wide variety of computer languages are available that have different use cases and strengths and weaknesses.

The Python and R languages are currently the most commonly used languages for data science, so we will focus on those languages in this course. We will also discuss Structured Query Language (SQL), which is used to query and manage databases and associated files. We will specifically explore SQL for querying data. The Bash language is also commonly used to interact with the terminal in UNIX and UNIX-like systems, such as Linux distros. However, we will not explore that language here.

Web development on client computers currently makes use of hypertext markup language (HTML) to define content, cascading style sheets (CSS) to define styles and responsiveness, and JavaScript to provide functionality. Other languages are commonly used on the server, such as PHP. We will not explore web development in this course.

It should be noted that data science is constantly evolving, and new languages are being developed. For example, the Julia and Rust languages are gaining ground in the world of data science.

IDEs

- Interactive Development Environment
- Application that makes programming easier
- Allows for code editing, syntax highlighting, autocomplete, building executables, debugging, viewing documentation, visualizing output, etc.

VS Code IDE

14

Interactive Development Environments (IDEs) are programs that make coding easier by assisting with code editing, providing syntax highlighting and autocompletion, supporting debugging, allowing viewing of documentation and help, and providing visual output.

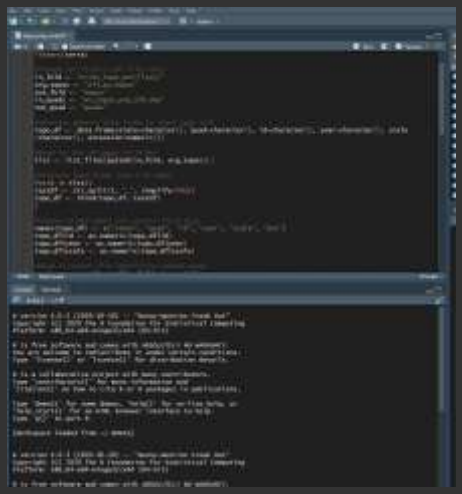
IDEs

Python

- Spyder https://www.spyder-ide.org/
- PyCharm https://www.jetbrains.com/pycharm/
- VS Code https://code.visualstudio.com/

R

- RStudio https://www.rstudio.com/



RStudio IDE

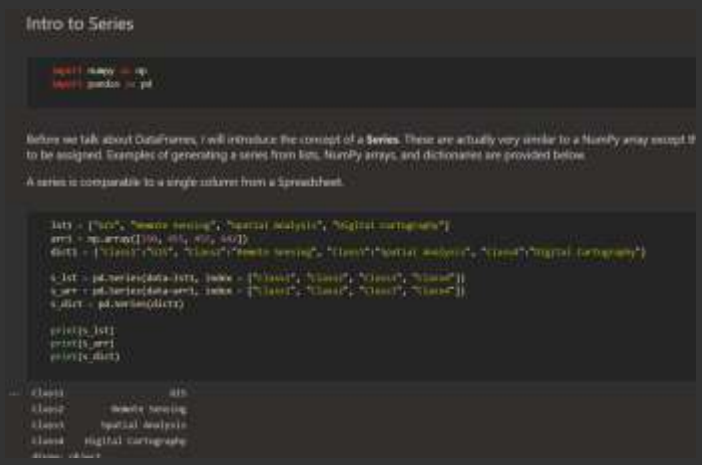
15

Example IDEs for Python include Spyder, PyCharm, and VS Code. In this course, I will make use of VS Code, which is freely available.

We will use RStudio as an IDE for R. RStudio is also free and open-source for single use. Note that it can also be used as and IDE for Python.

Notebooks

- Breaks code into cells
- Interactive execution
- Can include other types of cells (e.g., Markdown)
- Render results to PDF and HTML
- Jupyter Notebook
- JupyterLab



https://jupyter.org/

16

Another means to interactively develop code is to use Notebooks, as implemented with Jupyter Notebooks or JupyterLab. Notebooks allow for code to be partitioned into cells that can then be executed separately. Results, such as numeric results and graphs, can then be plotted in the flow of the page. It is also possible to include other types of cells, such as Markdown cells, which allow for inclusion of formatted text. Notebooks can even be rendered to Markdown documents, which can subsequently be converted to PDF documents or HTML webpages.

Jupyter Notebooks and the more recently introduced JupyterLab run within web browsers. However, Notebooks can also be used within some IDEs, such as VS Code and RStudio. Later modules will discuss how to set up and use Notebooks.

Command Line Shell

- Computer program that accepts keyboard/text commands as opposed to interacting with GUI
- Can be used to automate tasks
- Shell is accessed via the Terminal on MacOS or Linux distros
- Bash is a Unix shell and command line language often used in data science on Linux systems
- Bash shell can also be installed on Windows machines
- Scripts can also be executed on Windows machines using the PowerShell or Command Prompt

17

All operating systems provide command line shells. For example, Windows provides the PowerShell and Command Prompt. These shells commonly have their own language in which commands are executed. In UNIX and UNIX-like operating systems, shells are generally accessed using the terminal.

Bash is a UNIX shell and command line language often used within Linux distros.

Bash is used to run scripts and automate processes. We will not explore Bash in this course; however, it is worth investigating as you develop as a data scientist.

REPL

- Read-Evaluate-Print Loop
- Allows for working from the command line shell
- Read = Accept expression from user
- Evaluate = evaluate/execute expression
- Print = print result back to console
- Loop = repeat for next expression

18

Before moving on, I wanted to explain the concept of a REPL.

REPL stands for Read-Evaluate-Print Loop. In a command line shell, code is executed by (1) accepting or reading an expression provided by a script file or by manual input from the user at the command line, (2) evaluating the expression and executing the command, (3) printing results or messages back it the console, and (4) repeating the process as a loop to move on to the next expression.

The REPL method is used to execute code , expression-by-expression, at the command line or terminal and is a key component of how coding and scripting works.

Summary of Key Points

- Humans have developed several means to represent information.
- Computers make use of binary since information can be represented as on-and-off signals associated with transistors.
- Combining these on-and-off signals allow for the representation of more complex data and information.
- Coding is a key skill for data scientists as this is how we communicate with computers to perform key tasks. Being a data scientist requires some knowledge of coding.
- Many coding languages exist that have varying strengths and weaknesses. A few languages, such as Python and R, are currently used by data scientists.

Coding Components

20

The goal of this section is to introduce some key concepts and methods for coding and scripting in general. I will not explore a specific language or even show examples in specific languages. Instead, I will conceptualize key concepts.

Later modules will explore these concepts specific to coding languages commonly used in data science: Python and R.

I argue that understanding these key concepts is important prior to learning the specifics of a given language.

Objects and Classes

Vehicle

Properties	Methods
Make	.setMake()
Model	.setModel()
Engine	.setEngine()
Weight	.setWeight()
Gas Mileage	.setMileage
Cost	.setCost()

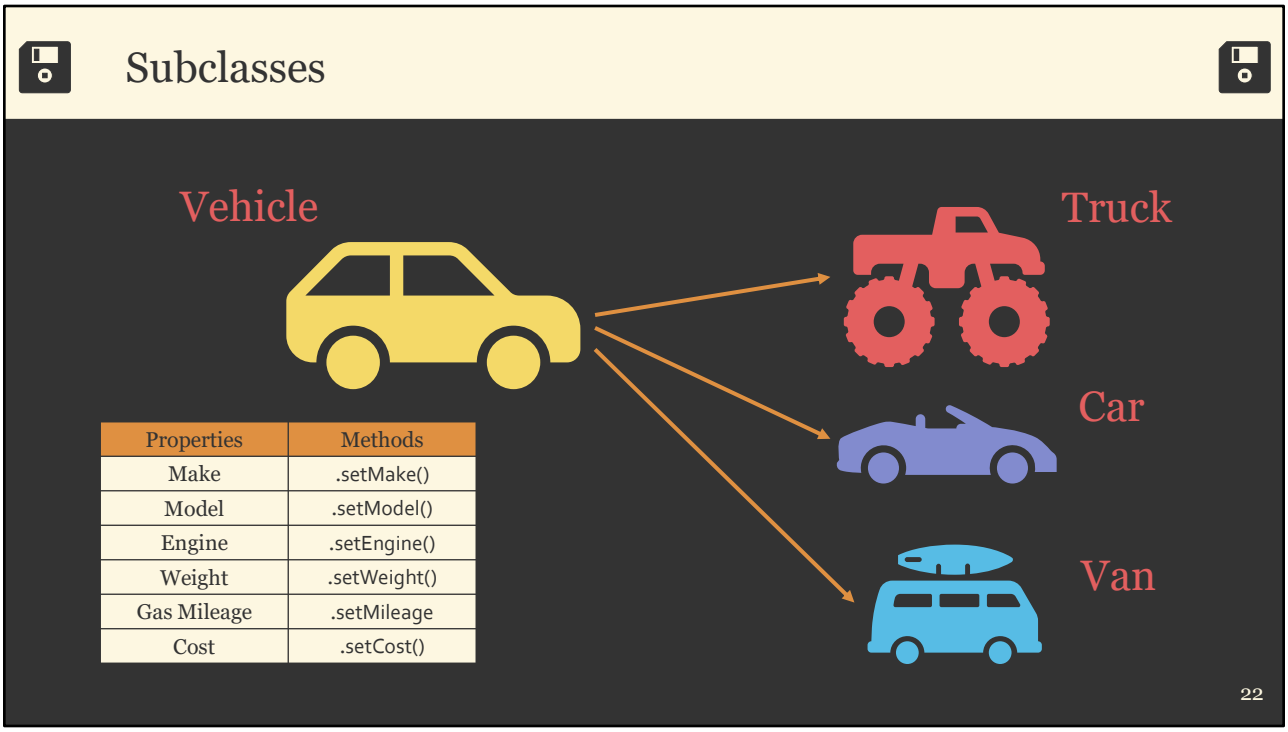
Tree

Properties	Methods
Genus	.setGenus()
Species	.setSpecies()
DBH	.setDBH()
Height	.setHeight()
Health	.setHealth()
Age	.setAge()

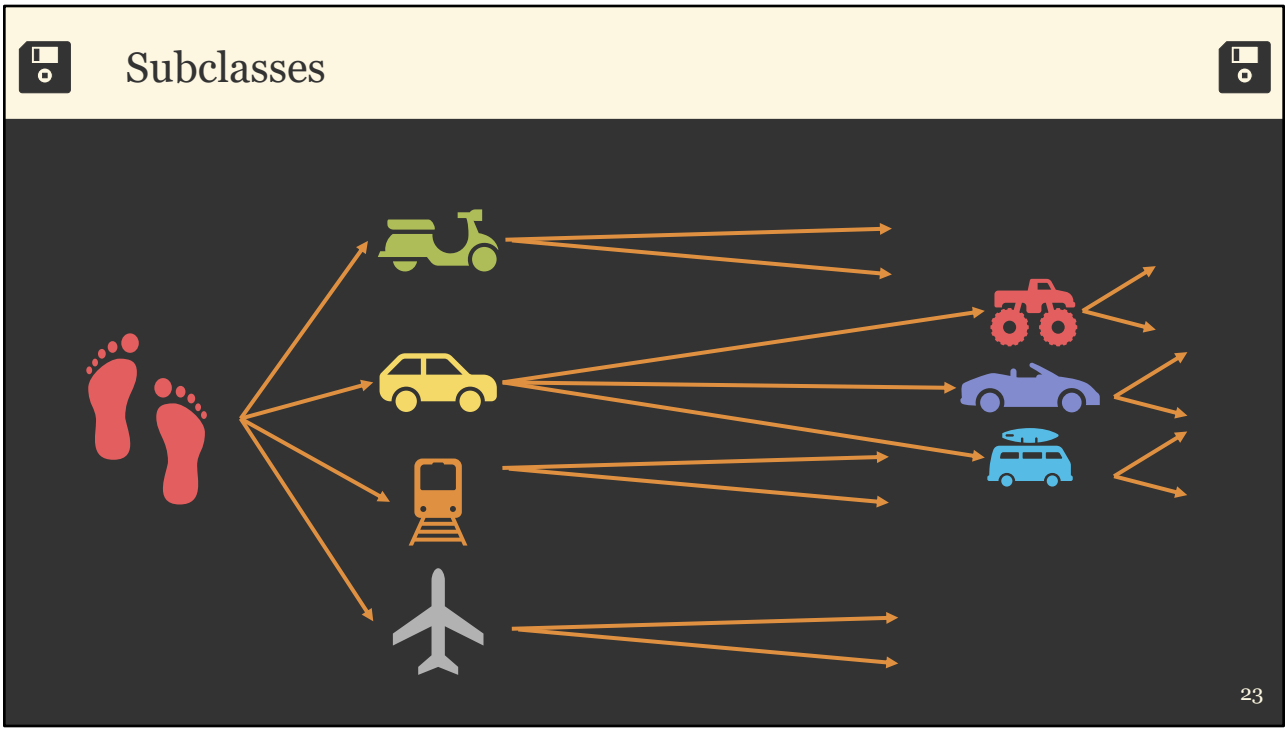
21

One key concept in coding is objects or classes. Objects or classes represent a certain type of real-world feature that is being modeled or represented in the computer. For example, you could define a class that is designed to represent video data or GPS points. The class will have specific properties that apply to it. For example, a class representing video data may have a property that indicates the number of frames per second or the resolution (HD vs. 4K.). There are also methods defined that can perform actions of some kind. For example, a method could print the run time of the video file. Methods are often described as functions tied to objects or classes. We will discuss functions later in this section.

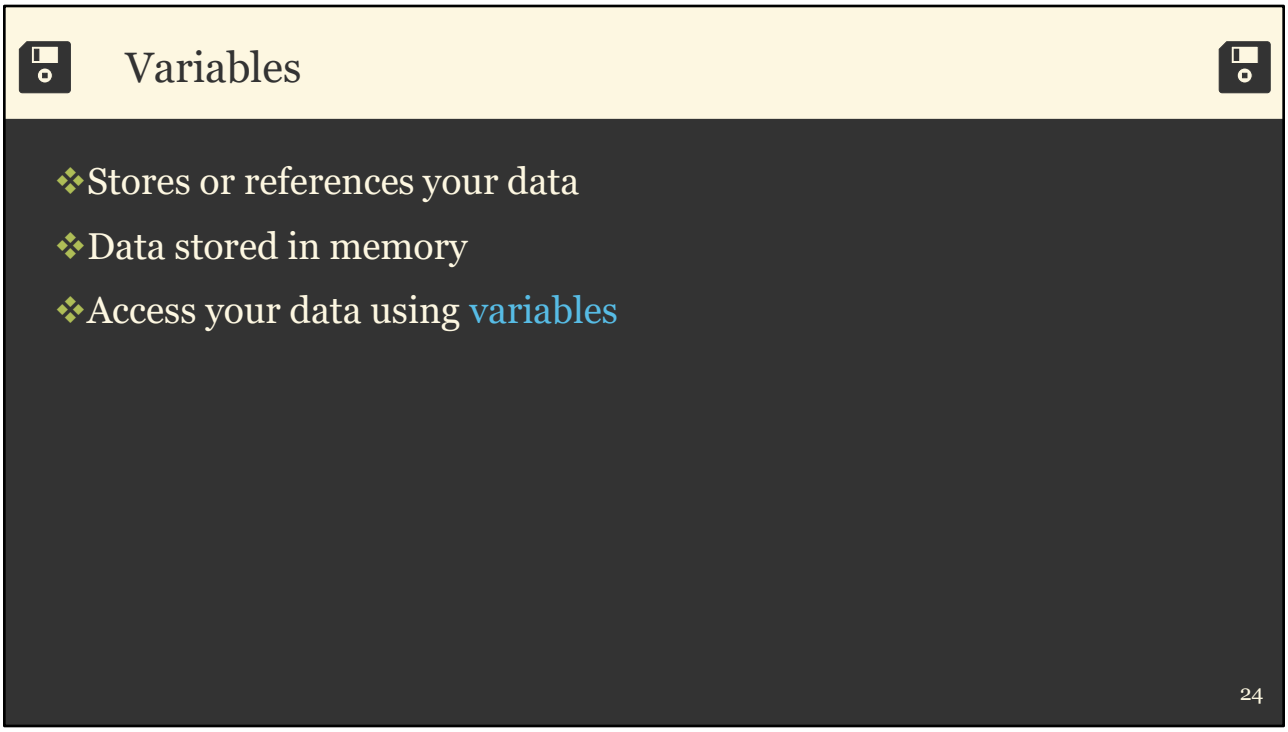
Classes commonly have constructor methods that are used to initiate an instance of the class, for example specific video data. The defined properties and methods can then be applied to this instance of the class.



Classes can be subclassed to create more specific representations related to the more general class from which they were derived. For example, a class representing vehicles could be subclassed to create classes representing specific types of vehicles. These subclasses can inherit properties and methods from their parent class, and new or altered properties and methods can be defined. For example, the truck subclass may have an added towing capacity property that is not applicable to the cars subclass.



It is even possible to subclass subclasses to make even more specific representations or models of real-world objects. For example, a transportation class could be subclassed into motorcycle, vehicle, train, and airplane classes. The vehicle subclass could then be subclassed again into truck, car, and van subclasses. With the creation of subclasses, now properties and methods could be generated specific to those types of features. Further, the more generic properties and methods from the parent class can be inherited.



In order to interact with our data using code, we must be able to reference them in the code. This is the purpose of variables. A data object stored in memory, or in RAM, can be referenced by assigning it to a variable. These variables can then be referenced within your code or provided as input to routines or functions in order to perform some operation or calculation that requires these data as input.

For example, a numeric value stored in memory can be associated with the variable x. The numeric variable can then be squared by applying a function or operation that squares numbers to the variable and its associated data.

Different programming languages have rules relating to how variables must be named (for example, it is common for variable names to not be able to start with a number). Also, different languages will have rules relating to if variable names can be reused or data associated with a variable can be altered or overwritten. These issues will be discussed when we talk about specific languages.

The key point here is that variables reference data and allow you to call or input data into processes defined by code and executed by your computer.

Data Types

- Numeric
  - Integer
  - Float
- Strings or Characters
- Booleans (True vs. False, On vs. Off, 1 vs. 0)
- Date/Time

25

All computer languages support several broad data types. Numeric data can be represented as integers, where no decimal values are stored, or float, where decimal values are stored. These data types can be used to represent interval or ratio data. However, sometimes numeric codes, generally integers, are used to represent nominal or ordinal data. However, in such cases, these values are just stand-ins for specific categories and should not be treated like numbers. For example, ZIP codes are used to designate certain geographic extents and should be treat as nominal data. It would not make sense to add or subtract zip codes.

Strings or characters store text. Such data are used to represent individual words or a series of words. They can also be used to represent nominal or ordinal data types. Numbers can be stored as text; however, they will be treated as if they represent words or characters as opposed to numeric values. So, it isn't possible to perform mathematical operations or arithmetic on numbers stored as strings.

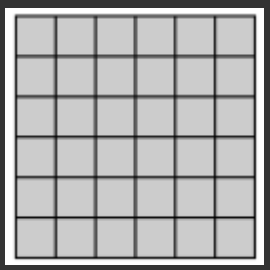
Booleans represent logically TRUE or logically FALSE. Note that this is different from strings. Booleans are often returned as a result of a test or query. For example, if you test whether 71 is greater than 50, the Boolean TRUE would be returned. In contrast, if you test whether 32 is greater than 50, the Boolean FALSE would be returned. TRUE can also be associated with the value

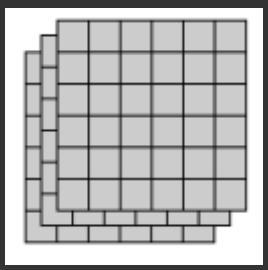
1 or on, in the context of binary data. In contrast, FALSE is associated with 0 or off.

Lastly, dates and times are often represented as a unique data type since unique operations are performed on dates. For example, you may want to determine the number of days between two dates, which is different from simply subtracting two numeric values.

Arrays

- Scalar = single value
- Vector = 1D array of values
- Matrix = 2D array of values
- XD Array = three or more dimensions





26

A series of values, characters or strings, or Booleans can be stored as a series. Although different names for such data structures are used in different programming languages, a common term is an array.

An array holding a single value is commonly referred two as a scalar. A series or list of values is commonly referred to as a vector.

Values stored in two dimensions, such as rows or columns, is an example of a matrix or two-dimensional array. An example of such data would be a black-and-white image in which each pixel in the image is associated with a specific row and column combination and holds a value that corresponds with a brightness (black-to-white). In contrast, a color image could be represented as a three-dimensional array in which the image is made up of a two-dimensional set of rows and columns of pixels, and each channel (red, green, and blue) is stored separately in a third dimension. In other words, the data are described or organized across three dimensions: rows, columns, and channels (or, height, width, and channels).

One of the useful characteristics of arrays is that a wide variety of data types can be stored by adding or removing dimensions as needed. For example, a video could be represented using four-dimensions: rows of pixels, columns of pixels, channels (red, green, and blue), and time (or frame number).

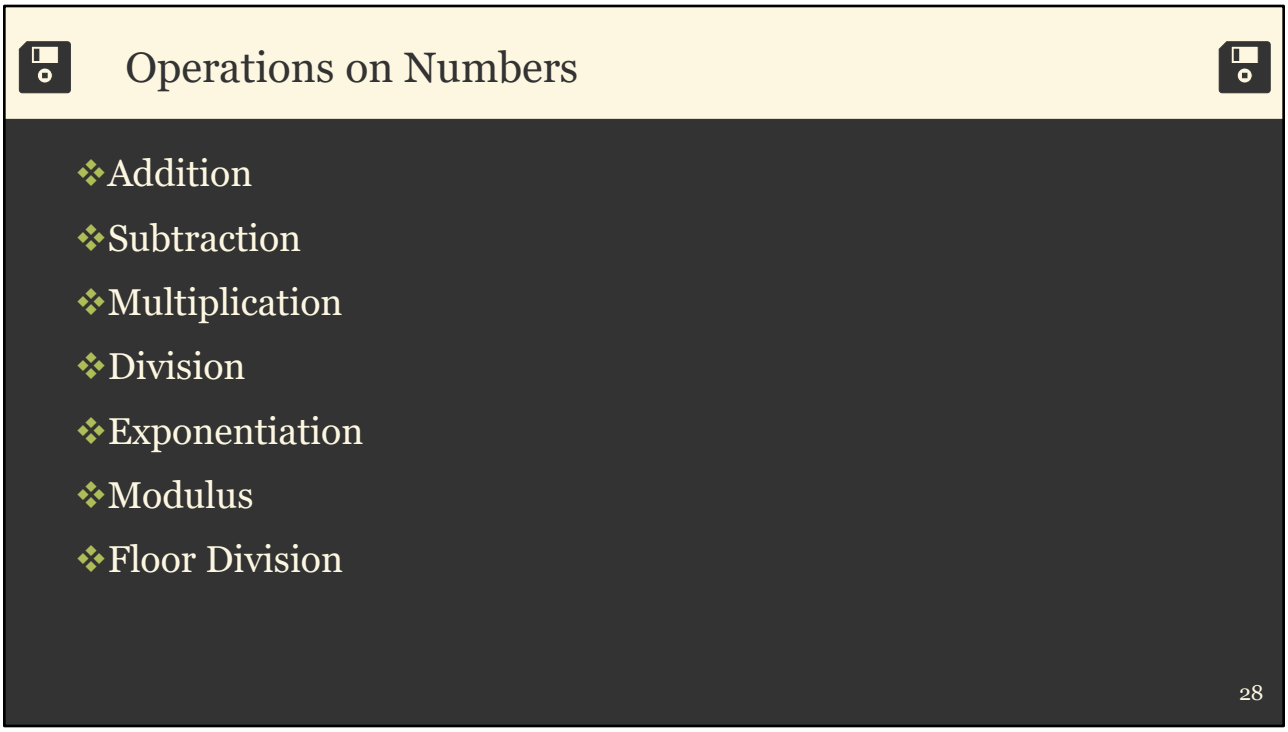
Since the array data model is so flexible and can be used to represent a variety of data, it is very important in data science.

Changing Data Types

- Integer ←→ Float (1 ←→ 1./1.0)
- Numeric ←→ String (1 ←→ "1")
- Numeric ←→ Boolean (1 ←→ TRUE)
- String ←→ Boolean ("TRUE" ←→ TRUE)

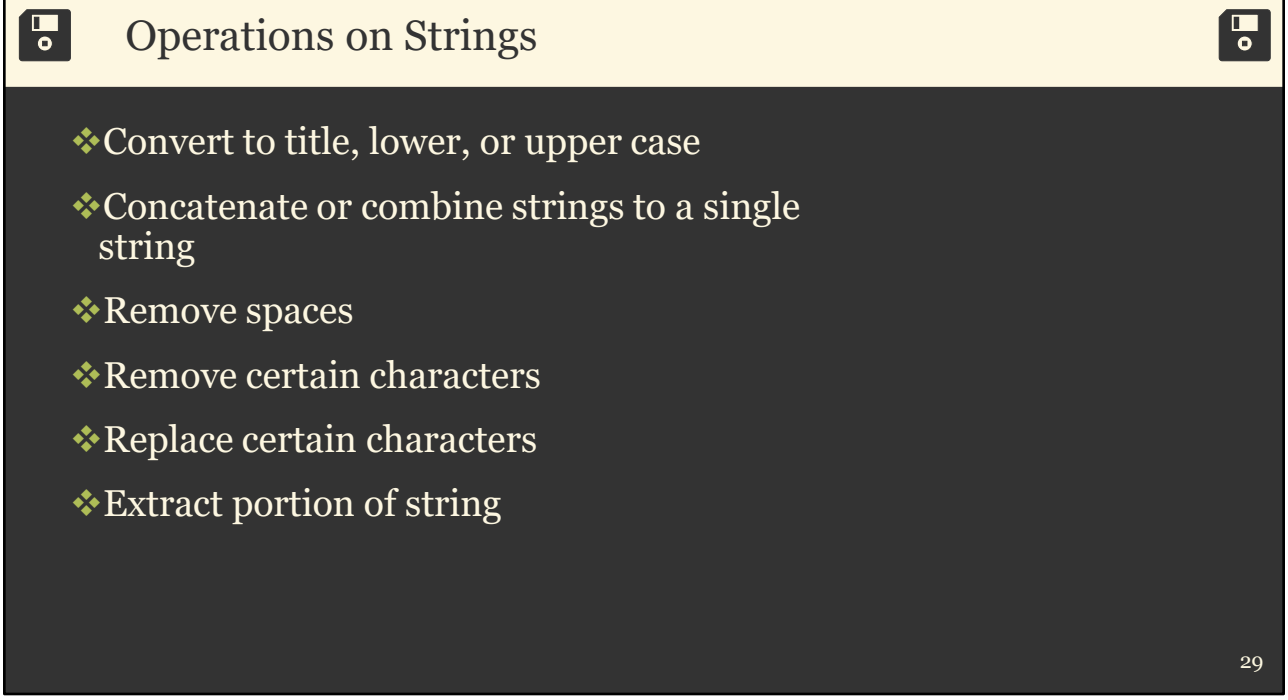
27

It is also generally possible to convert between data types. For example, numeric data can be converted between integer and float or converted to a character or string to be treated as text as opposed to a value. If a numeric value is converted to Boolean, 1s will generally convert to TRUE while 0s will convert to FALSE. It is also possible to convert "TRUE" and "FALSE" as strings to TRUE and FALSE as Booleans.

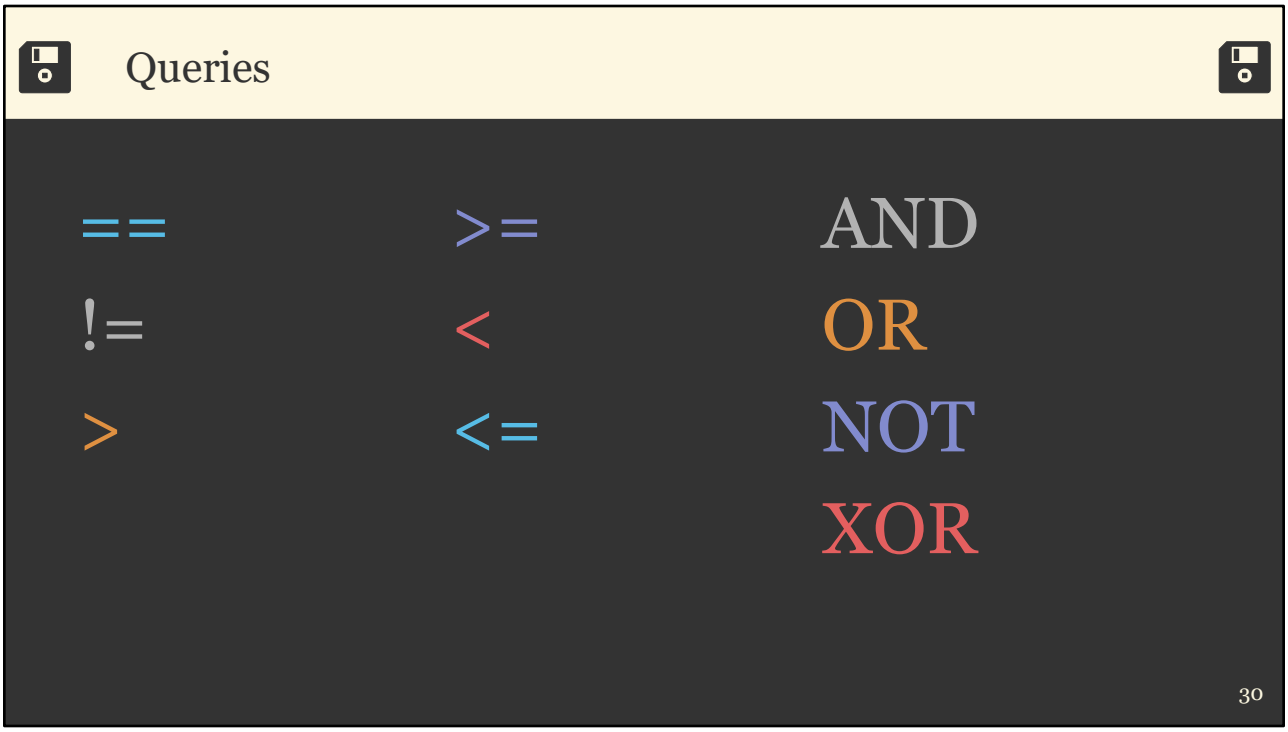


Programming languages support a variety of mathematical operations that can be performed on integer or float data. However, generally, integers often return only integer results, so will be rounded off. You may want to convert integer data to float prior to performing mathematical operations.

Modulus is the remainder after division. For example, when 4 is divided by 3, the modules would be 1. Floor division means to round down to the nearest whole number after division. For example, 10 divided by 3 would return 3.



It is also generally possible to perform a variety of operations on strings or characters. Examples include converting case, merging (or concatenating) multiple strings into a single string, removing spaces, removing certain characters, replacing certain characters, and extracting portions of strings.

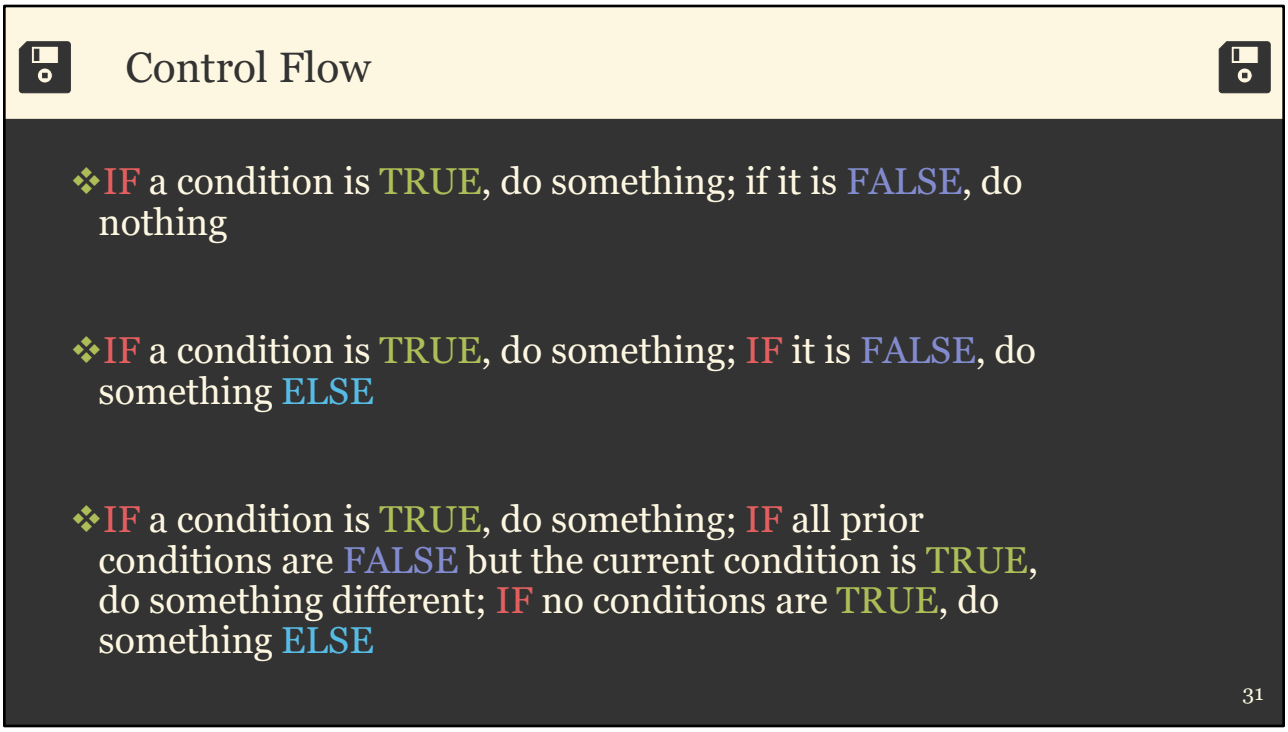


Queries can be used to return a subset of data that meet a certain criteria or as a test, which will return a Boolean.

For example, greater than could be used to return all values greater than a defined value or to test whether a specific value is larger than a specified value.

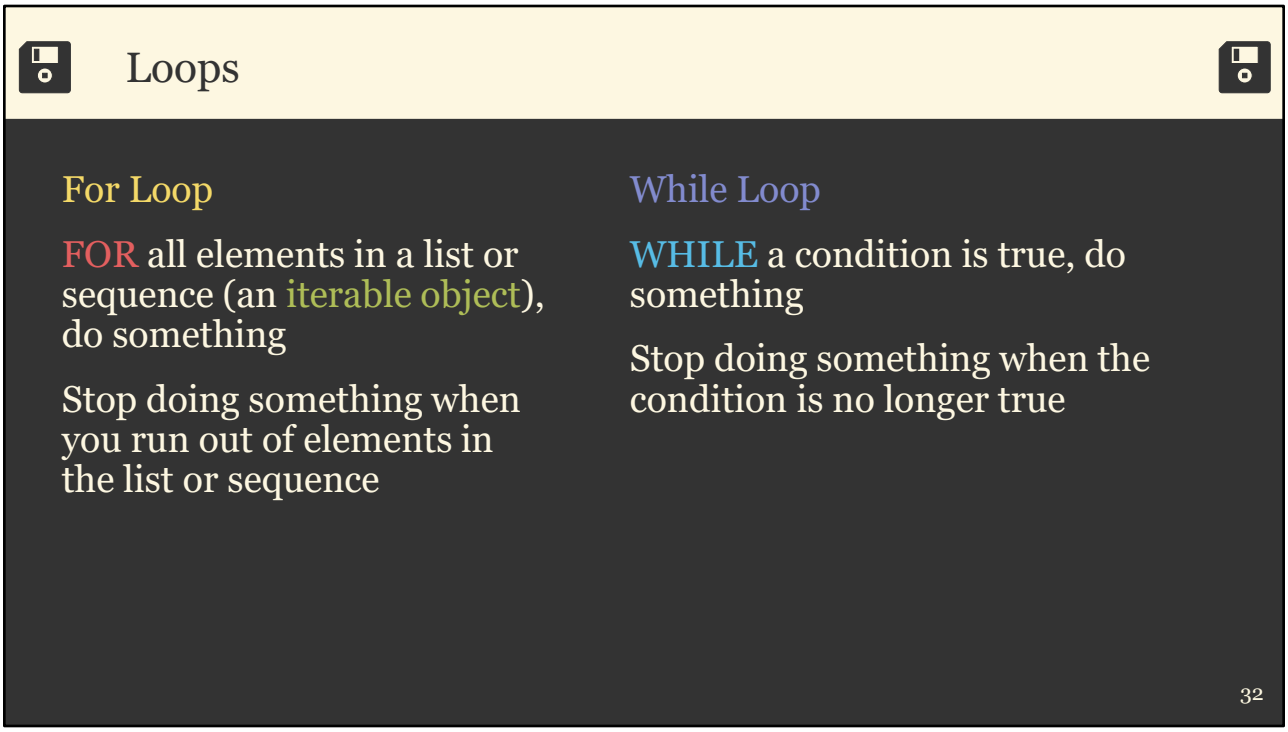
Compound queries can be generated using AND, OR, NOT, and/or XOR. AND indicates that both criteria must be met, while OR indicates that both or one of the two criteria must be met. NOT indicates that one criteria must be met but not the other. Lastly, XOR indicates that one of two criteria must be met, but not both.

When creating complex queries, it is often necessary to use parenthesis to deal with ambiguity in logic.



Control flow allows for different outcomes, or no outcome, based on a condition. For example, a list of employees could be sorted into two lists depending on whether or not the employee has worked at the company for at least three years. Further, a different operation could be performed depending on the condition. For example, employees that have been at the company for more than three years could be assigned a larger bonus than those that have been employed for a shorter period. So, the outcome or calculation can vary by case based on a condition.

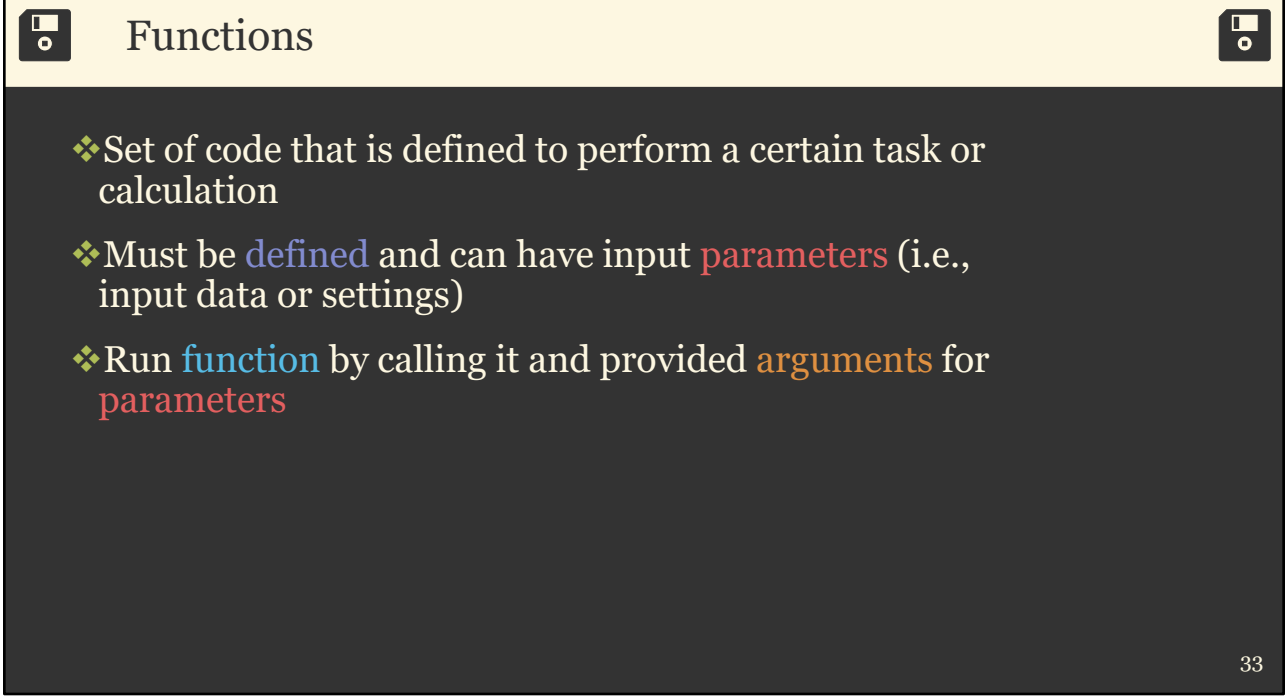
Control flow is generally accomplished using IF…Else statements. If the condition associated with the IF statement is TRUE, then certain code will be executed. If the condition evaluates to FALSE, then no code can be executed or code associated with an ELSE statement can be executed. If you want to include multiple conditions, you can incorporate additional IF statements (generally termed elif). The code associated with the first IF statement that evaluates to TRUE will be executed. If no IF statements evaluate to TRUE, then nothing will execute, or the code associated with the ELSE statement will execute.



Loops are used for iteration, or repeatedly running the same code with new inputs.

Specifically, a For Loop will execute code for each element in an iterable object, such as each number in a series or each element in a list or array. For example, in each iteration of a For Loop over a list of files, one of the files could be read in, processed in some way, and the final result could be written to disk. Once all the files have been processed, then the code will exit the loop. Further, combining a For Loop and an IF…ELSE statement can allow for different code to be executed during each iteration of the loop depending on a condition. For example, a statement could test whether a file is an image or an audio file. If it is an image file, some processing would be applied, while different processing would be applied to an audio file.

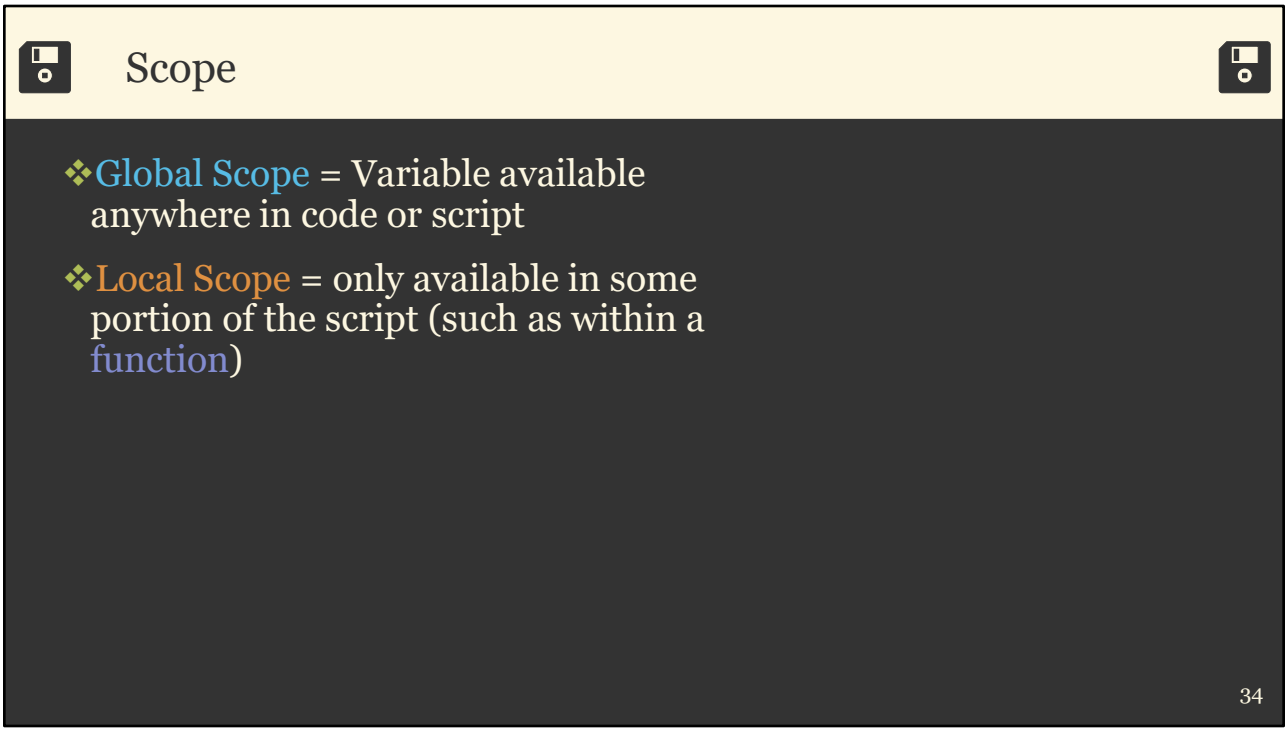
While Loops will continue to execute until a condition evaluates to FALSE. For example, a series of numbers could be processed until a certain value is reached. At that point, the code would exit the While Loop. If the condition never evaluates to FALSE, the loop may continue indefinitely, which is known as an infinite loop. Infinite loops are an issue that can arise from an incorrectly defined While Loop.



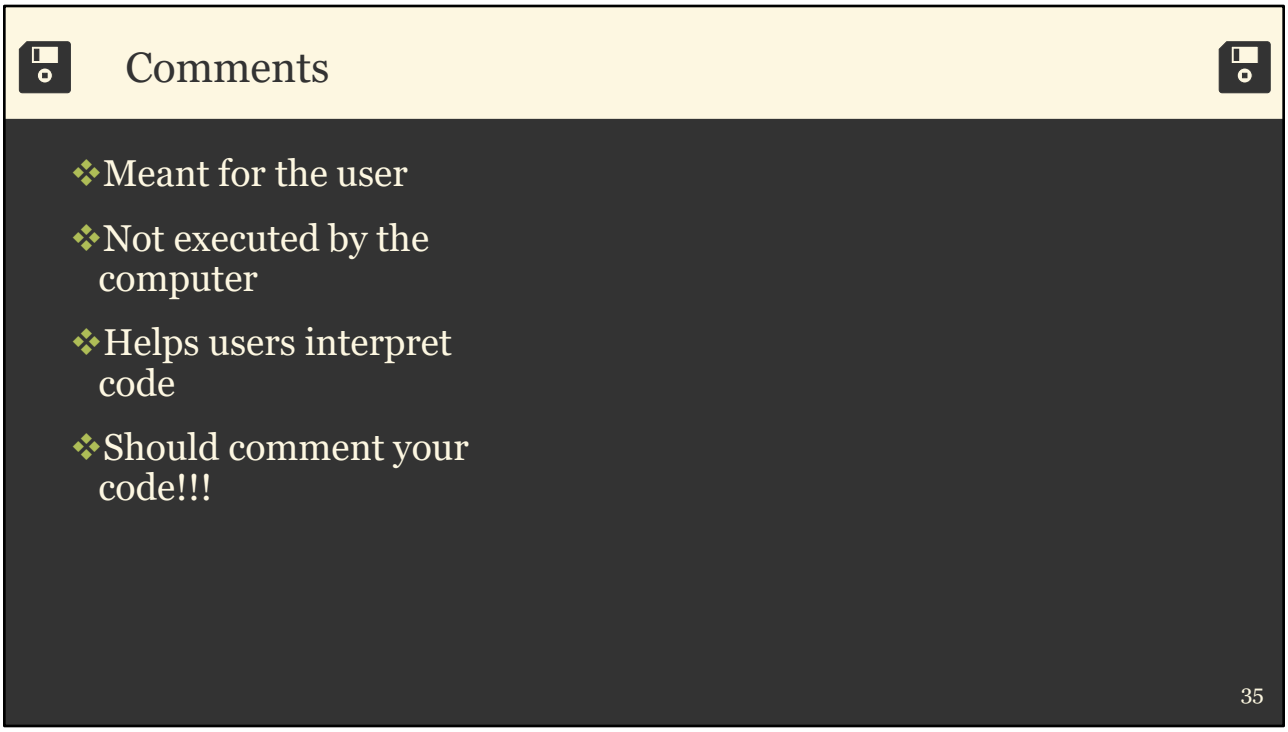
Functions are sets of code that can be repeatedly used to perform a task. They can make your code much more concise and help you minimize redundant code. All coding languages will have built in functions for common tasks along with additional functions that are made available via add-ons, libraries, modules, or packages.

Alternatively, you can define your own functions. When a function is defined, it can be set up with certain parameters that will accept arguments. For example, you could define a function that will multiply two values together. Such a function would require that two values be provided as arguments.

Functions are a key part of all programming languages and are used extensively in data science.



The concept of scope or namespace is important when coding. Generally, variables that are defined outside of a function have global scope. Or, they can be called anywhere in the code, even within functions. In contrast, variable defined within a function generally have local scope, or they can only be used or called within the function in which they are defined.



Comments are meant for the user or human as opposed to the computer. Comments are added to make your code more interpretable for later use or for use by others. Different languages will use different syntax to denote a comment. Both Python and R use the hashtag or pound symbol to denote a comment. When the computer executes the code, comments will be ignored.

It is good practice to comment your code.

Summary of Key Points

- Although all coding languages are different, the do share some common attributes.
- Within a computer language, abstract models can be generated to represent specific types of data.
- Data can be referenced using variables within code.
- Different types of data can be represented and associated with a variable including numbers, text strings, Booleans, dates, and arrays.
- A variety of operations can be performed on data via coding.
- Flow control allows for different operations to be performed, or no operation to be performed, based on conditions.
- Loops allow for iteration, or the repeated execution of code, on varying data inputs.
- Functions are sections of code that are defined to perform a certain task.

This is the end of this lecture module.

Please return to the West Virginia View Webpage for additional content.

Thanks! Hope you found this useful.