QUESTION 1

MARKUP

Web development markup languages are the core building blocks utilised in constructing web pages. The most important concept that defines these languages is 'Markup'. Markup predates computers; it can be comments left on a draft document, paragraphs being highlighted in textbooks or annotations on a report (Kyrnin, 2021).

Markup can be considered as instructions indicating that certain text needs to be manipulated; highlighted or de-emphasized. Web development markup languages use markup to instruct the computer on how the text should be formatted for display (Varagouli, 2021). HyperText Markup Language (HTML), eXtensible Markup Language (XML), and eXtended HyperText Markup Language (XHTML) are three extensively used web markup languages (Kyrnin, 2021).

TAGS & ELEMENTS

Tags are a component that is widely seen in web markup languages. A tag consists of less-than '<' and greater-than '>' symbols to signal that it is part of the markup. Tags are used in pairs to bookend an element, with the second tag including a back slash to indicate the element's end (Christensson, 2006). A string contained within the tag indicates how the element will be formatted. For example, tags used to display a table would look like this:

```
<table> </table>
```

Tags are important because they express the contents of the element (semantic markup) as well as how it will be presented (presentational markup) (Varagouli, 2021). An element comprises a beginning tag, content and an ending tag. For example, a title element looks like this:

```
<title> This is a Title for a Web Article </title>
```

SEMANTIC & PRESENTATIONAL MARKUP

When a tag explains the purpose of the element's contents, it is considered semantic markup. For example, contents within a <title> tag is likely to be the page's title. And content in a <nav> tag is likely to be used as a way to navigate the page (Varagouli, 2021).

Presentational markup is used to describe the visual appearance of content. This includes usage of <i> and <b> tags to italicise text and bold text respectively. Though over time this type of markup has been deprecated in favour of Cascading Style Sheets (CSS), as there became a need for the separation of the contents of the page and its styling (Kyrnin, 2021).

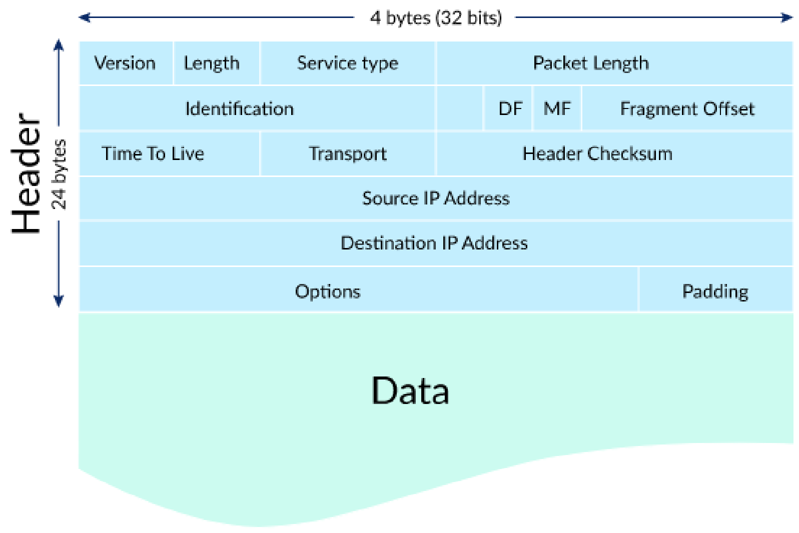
QUESTION 2

PACKETS

The Internet is essentially a vast network of computers communicating with one another and exchanging data, and all of this data is transmitted in a series of packets. Whenever any data is sent over the internet, be it an image, email or webpage; it will essentially be broken down into small packets of data, and sent to its destination using the best available route. When all the packets arrive at their destination, they will be reassembled to form the original piece of data (HowStuffWorks.com, 2021).

The use of packets was important to the development of the internet as sending data directly between computer devices is greatly impractical. Sending larger forms of data such as images or videos without the use of packets would be pushing the physical limitations of the network. Not only this, but having two computers exclusively sending data to one another would prevent other computers from using the route to send or receive information; they would have to wait their turn. Packets circumvent these issues as they can be sent to their destination across any route across the internet, and in any order (in certain protocols), allowing billions of computers to exchange data over the same network, concurrently (Cloudflare, n.d.).

The structure of a packet can be broken down into its Header, Payload and Trailer. The header generally contains information about the packet (number in sequence, protocol, length), as well as its originating and destination addresses. Routers will read the information in the header and know where to send the packet. The payload is the actual data that the packet carries. And the trailer generally indicates to receiving devices that they've reached the end of the packet (HowStuffWorks.com, 2021).



Data Packet Header structure example.
From Khan Academy (Fox, n.d.)

IP ADDRESSES (IPv4 and IPv6)

Internet Protocol (IP) was core to the development of the internet as it handles the addresses of devices connected to the internet, as well as the routing of data packets to those addresses. Put simply, IP addresses are similar to mailing addresses for a computer connected to the internet. As was previously established, data packets need to have a recipient IP address as well as a sender IP address. Today, there are two versions of IP addresses in use; IPv4 and IPv6 (Fox, 2020).

```
// An example IPv4 address
74.125.20.113
```

Example IPv4 address.
From Khan Academy (Fox, 2020)

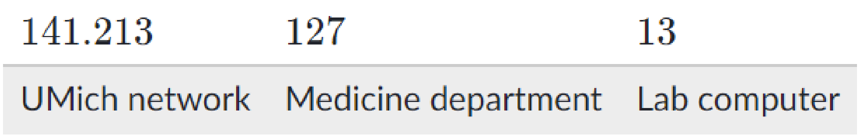
The first version of Internet Protocol to be used on the internet was IPv4, and as seen above, it consists of 4 numbers separated by a period. Though the numbers are displayed as decimal, they are stored in binary, and therefore are limited to only 8 bits. Therefore each number or 'octet' can only range from 0 - 255. And with only 4 numbers to an IPv4 address, you quickly reach a ceiling of only 4,294,967,296 possible IPv4 addresses (Fox, 2020). Though the number may seem large, with the increasing number of devices connecting to the internet today, a new solution was needed.

```
// An example IPv6 address
2001:0db8:0000:0042:0000:8a2e:0370:7334
```

Example IPv6 address.
From Khan Academy (Fox, 2020)

Enter IPv6; where values are stored in 8 hexadecimal numbers instead of the 4 decimal numbers with IPv4. Each 4 digit hexadecimal number can range from 0 - 65,535, and with 8 numbers, it results in over 340 undecillion possible IPv6 addresses (Fox, 2020).

The numbers in an IP address not only serve as a unique identifier, they are also ordered in a hierarchy to aid in routing data on the Internet and to improve efficiency of data flow. For example, a phone number can have a country code, area code, local exchange and then your specific number trailing at the end. IP addresses work in the same manner, where the first 2 octets may represent a network provider or Internet Service Provider. The following octets will identify a local network or device within the larger network. This hierarchical scheme results in packets being sent to a network administrator, where a more local router can then manage sending the packet to the computer (Fox, 2020).

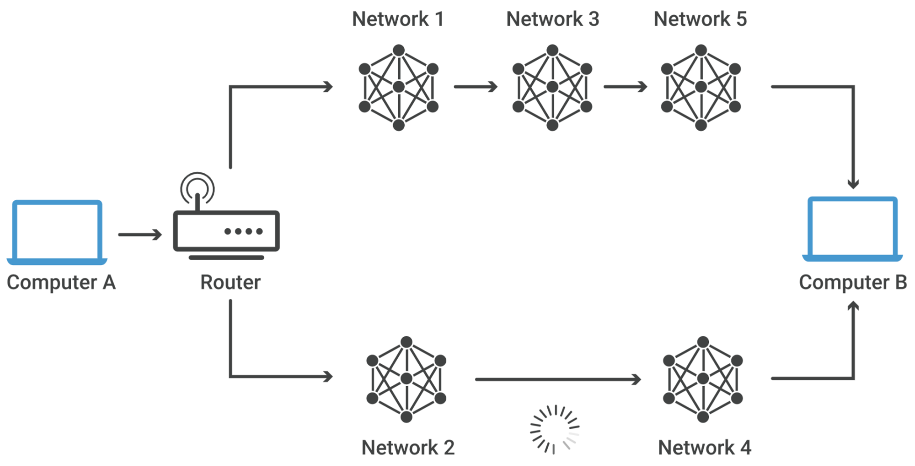


141.213	127	13
UMich network	Medicine department	Lab computer

Hierarchical Example of an IP address (141.213.127.13)
From Khan Academy (Fox, 2020)

ROUTERS & ROUTING
As previously established, data is transferred between computers through the Internet using packets, each with an IP address of both the sender and the recipient. Network routing determines the best route for these packets to follow from their origin to their destination. Similar to telephone networks or public transport, finding an efficient route to your destination is important, and in the case of internet routing, is handled by hardware called routers (Cloudflare, n.d.).

When a router receives a packet, it reads the destination address and uses an internal routing table to identify the best network path to take. In the same way that a train conductor would read the train schedule and direct a commuter on which train to take to get to their destination. Packets can be sent to multiple routers before they reach their destination, each router processing millions of packets per second (Cloudflare, n.d.). Routing tables can be static, in that the routes used are set up by an administrator and do not change. Or they are dynamic, where different protocols are used to find the quickest route to the destination.



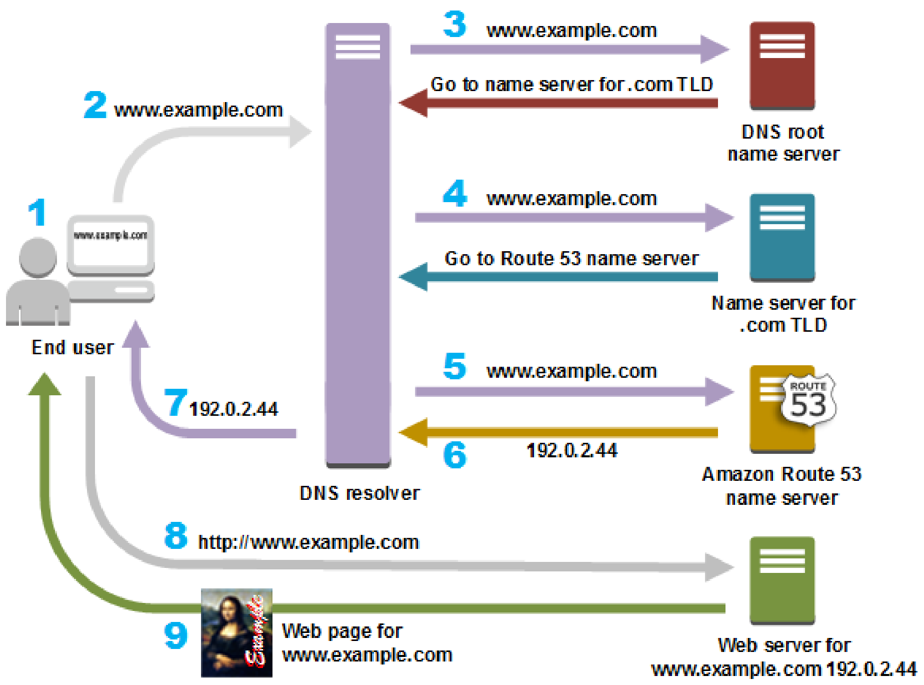
Possible routes for packets to be sent through.
From Cloudflare "What is Routing?" (Cloudflare.)

For example in the diagram above, the router has to send packets originating from Computer A to their destination at Computer B. While going through Network 2 and 4 seems the shortest path, it would actually take longer as they are congested with other packets that need to be processed first. So the router determines that sending packets through Networks 1, 3 and 5 to be the quickest route (Cloudflare, n.d.). Routing protocols such as Open Shortest Path First (OSPF) are used to identify the shortest and fastest route. Routing Information Protocol (RIP) is also commonly used, where the most desirable path is one with the least amount of 'hops' (each hop being when a packet gets sent from one network to another) (Cloudflare, n.d.).

DOMAINS & DNS

Domains and Domain Name Systems (DNS) have been fundamental to the development of the Internet, as they are what makes it possible to browse the World Wide Web with ease. As stated previously, devices connected to the Internet all have an IP address consisting of a string of numbers. Meaning that all websites have an IP address which is what computers use to access their content. But remembering the IP address of hundreds of websites is difficult for users, so we use domain names like "google.com" instead. It is the DNS protocol that helps convert the domain name into an IP address that the computer can use to retrieve the webpage (Brain et al., 2021).

In essence, DNS can be considered a phonebook for the internet. When a user enters a domain name into their browser, a DNS server essentially checks through a database to find the IP address that belongs to the requested domain, this is known as DNS name resolution. There are two types of DNS services, Authoritative DNS and Recursive DNS. Authoritative DNS has the final authority over domain names and responds to Recursive DNS servers with the IP address information. Recursive DNS, also known as a Resolver, is what a user's computer will mostly be communicating with. The Resolver doesn't own any DNS records, and instead acts as an intermediary that gets information on behalf of the user. When a computer sends the Resolver a domain, it communicates with higher level name servers until it finds the respective IP address. It then saves this information in it's memory cache for future reference, before passing it back to the user computer (Amazon AWS, n.d.).



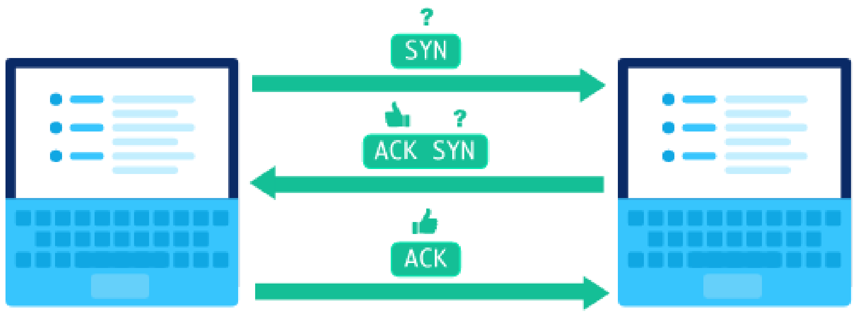
How Amazon DNS Servers function.

From Amazon AWS Documentation (Amazon AWS, n.d.)

QUESTION 3

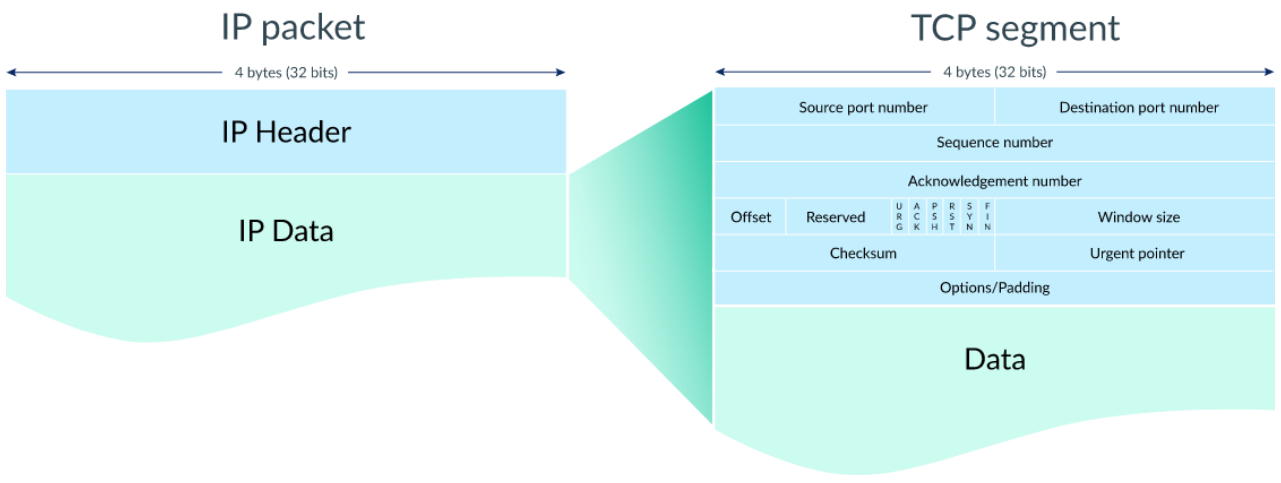
TCP

Transmission Control Protocol (TCP) is essential to the functioning of the Internet as it defines how computers connect and send data packets to each other over a network. It is a connection-oriented protocol, meaning that a connection is established between two computers, and maintained until all data has been exchanged (Lutkevich, 2021). TCP also ensures that no data packets are lost, jumbled or received twice over a connection, which is important in maintaining data integrity. A connection between computers under TCP is established using a three-way handshake.



A three-way handshake used by TCP
From Khan Academy (Fox, 2020)

In this handshake, the first computer sends a packet asking to connect ("synchronise"). The second computer sends back a packet acknowledging the first, and also asking to synchronise. The first computer then sends the third and final packet acknowledging the second's sync request. The connection has now been established and data transfer can begin (Fox, 2020). These synchronise and acknowledgement requests are stored in the packet header which can be seen below.



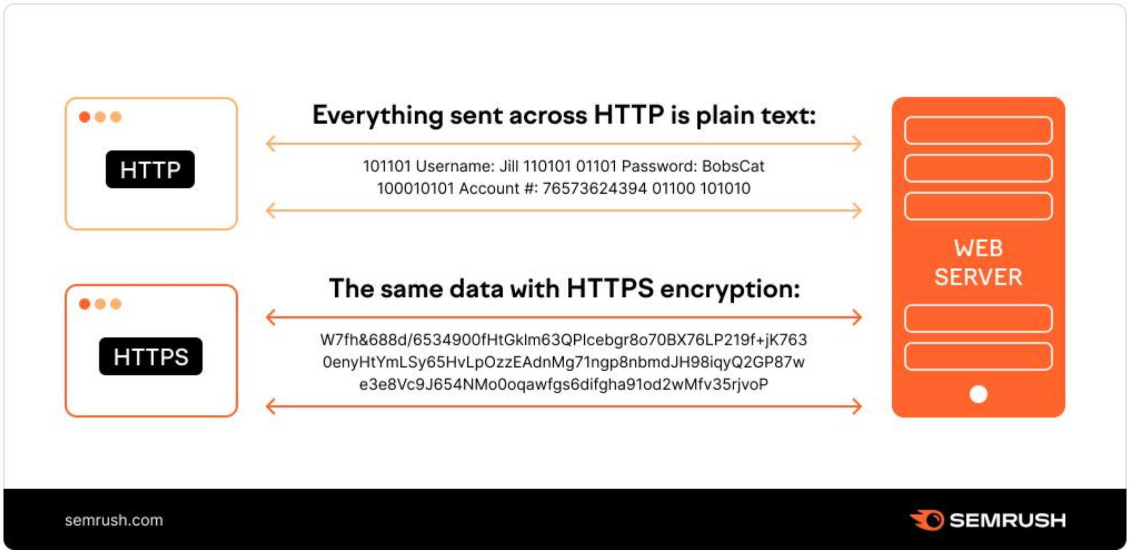
TCP Segment in an IP Packet Header
From Khan Academy (Fox, 2020)

When sending data packets over a network, they can be lost, duplicated or arrive at their destination out of sequence, (remembering that packets can all take different routes to their destination). TCP is responsible for ensuring that all data packets are received and in sequential order. The sequence number seen in the packet header allows the receiving computer to reassemble the data packets in the proper order. The receiving computer acknowledges every packet it receives, and any packets not acknowledged will be sent again. When all packets arrive and have been acknowledged, the receiving computer can discard any duplicate packets and reassemble the data in sequential order

(Fox, 2020). The security and integrity that TCP provides has been essential to the development of the Internet, and allows transmission of data over higher level protocols such as Secure Shell (SSH), File Transfer Protocol (FTP), Mail transfer protocols (SMTP, POP & IMAP) and even HTTP for website access (Lutkevich, 2021).

HTTP & HTTPS

Hypertext Transfer Protocol has been crucial to the development of the Internet, and without it, there would be no World Wide Web. The main purpose of HTTP is to process, render and deliver web pages to your computer from a web server; it does this with the use of HTTP requests (Harnish, 2020). A GET request is how web browsers ask for the information needed to display a website, whereas a POST request typically is used to submit information to the web server; a user's login details for example (Cloudflare, n.d.). This is where issues arise, as data sent through HTTP is clear and unencrypted, so any information that is transmitted can be intercepted and read by third parties.


HTTP
Everything sent across HTTP is plain text:
101101 Username: Jill 110101 01101 Password: BobsCat
100010101 Account #: 76573624394 01100 101010
HTTPS
The same data with HTTPS encryption:
W7fh&688d/6534900fHtGklm63QPlcebgr8o70BX76LP219f+jK763
0enyHtYmLSy65HvLpOzzEAdnMg71ngp8nbmdJH98iqyQ2GP87w
e3e8Vc9J654NMo0oqawfgs6difgha91od2wMfv35rjvoP
WEB SERVER
semrush.com
SEMRUSH

Data transmission using HTTP and HTTPS
From Semrush Blog (Harnish, 2020)

This is where Hypertext Transfer Protocol Secure (HTTPS) becomes incredibly useful. HTTPS is a secure version of HTTP, using the Transport Layer Security (TLS) or Secure Sockets Layer (SSL) encryption protocols to keep information safe. With HTTPS, first a secure connection is established using a SSL/TLS Handshake (somewhat similar to a TCP three-way handshake), and then data is encrypted and decrypted using an asymmetric public key infrastructure. Essentially, any sensitive data that a user sends to a web server will be jumbled using the public key, sent to the web server where it will be decrypted using the private key. Even if this data was intercepted on its way to the web server, the third party will only receive unreadable encrypted data, as seen in the diagram above (Harnish, 2020). HTTPS allows us to safely and securely send and receive sensitive data, making possible services such as internet banking, shopping and communications.

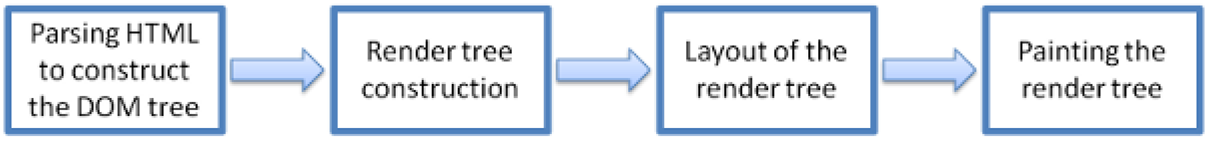
WEB BROWSERS

Web browsers are possibly the most widely used software on computer devices. They are used to access web pages on the World Wide Web, and they do this by requesting the resources from a web server and displaying them in the browser window. The first step when a user enters a website URL in the address bar is to make contact with the DNS server and find the relevant IP address the browser has to connect to. Once identified, the browser initiates a TCP connection with the web server and begins a HTTP exchange; requesting the data for the web page (Garsiel & Irish, 2011).

```
GET / HTTP/1.1
Host: google.com
Accept: */*
```

Example browser request for www.google.com
(Nadalin, 2018)

The web server will respond to this request, typically with some HTTP protocol headers, but also the main piece of data contained in the body of the response: the HTML document (or in some cases, PDF, XML, etc.) Though the HTML code will be unreadable to your average user in its text format, and so it is also the browsers role to render the code for display on the users screen. The browser parses the HTML markup, loads any additional resources such as CSS and Javascript, and presents it on the users screen (Nadalin, 2018). There are various web browsers currently being used, and they do not all use the same rendering engine, meaning that the same website code can look different when rendered by different browsers.


Parsing HTML to construct the DOM tree
Render tree construction
Layout of the render tree
Painting the render tree

Rendering Engine Basic Flow
(Garsiel & Irish, 2011)

A simplified portrayal of the rendering process can be seen in the diagram above. Firstly, a Document Object Model (DOM) Tree is constructed using just the HTML. Then, styling is applied to each node of the DOM Tree using CSS and inline styling, this is the Render Tree. The Render tree then goes through the layout process, where each node will be given screen space coordinates. And finally, the Render Tree is painted onto the screen. This entire process is gradual for a better user experience, meaning that parts of the web page will appear while other sections are still being delivered over the network (Garsiel & Irish, 2011).

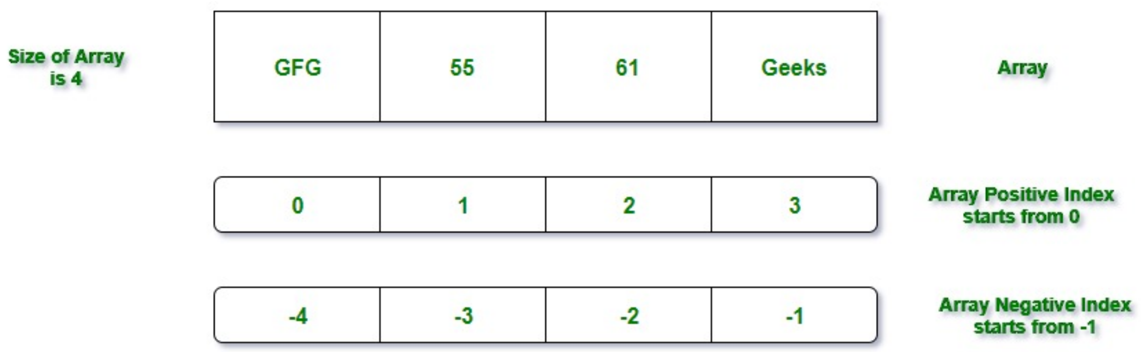
Web browsers are integral to the Internet as they allow the average user to view web pages without having to fiddle with any web code or communication with a web server. However, many browsers do provide developer tools that give more advanced users access to the process of building web pages. The Document Object Model, which takes a website's HTML source code and represents it as a node tree, can be manipulated using developer tools. Not only this but CSS and Javascript can be inspected and manipulated on a local level with the use of developer tools (Knowles, 2020). Developer tools allow developers and advanced users to debug and study the processes involved in the delivery and display of web pages.

QUESTION 4

ARRAYS

Arrays are a collection of indexed objects stored at contiguous memory locations, meaning that objects are stored one after another without any gaps in between (Saini et al., 2021). In Ruby, the stored objects can be integers, symbols, strings and even other data structures such as hashes and arrays. Ruby also manages the computer's memory when the size of the array expands and shrinks, unlike other lower-level programming languages (Castello, 2019).

```
example_array = ["GFG", 55, 61, "Geeks"]
```


Size of Array is 4
GFG
55
61
Geeks
Array
0
1
2
3
Array Positive Index starts from 0
-4
-3
-2
-1
Array Negative Index starts from -1

Creating an Array using a literal constructor [ ]
From GeeksforGeeks.com (Saini et al., 2021)

In the example array above, elements are added to the array by listing them, separated by commas and surrounded by square brackets. And as seen in the diagram, the resulting array has the elements placed in an indexed collection. If the user wanted to retrieve data from the middle of the array, they could access it using the corresponding index (Saini et al., 2021).

Arrays can be quite flexible and have many use cases. They can be used in the most basic way as an unordered list of data, the names of students in a classroom, or a shopping list for example. Arrays can also be sorted into a sequence if there is a need for sequential data, and the items in an array can also be iterated through: running code for each item in the array. Having an array of arrays, or a 2-dimensional array can also be very useful, allowing you to store data in a matrix-like format. For example, if you were to make a game using a deck of playing cards, each suit could be its own array, with all 4 suits in a deck array. You could then go to a random card index, within a random suit index to pull a random card!

HASHES

One of the most powerful and useful data structure types in Ruby is the hash. Unlike the array which stores data indexed using integers, the hash stores its data against unique keys, often referred to as key-value pairs (Castello, 2020). This allows data to be represented associatively, with each unique key having a value connected to it as seen in the example following.

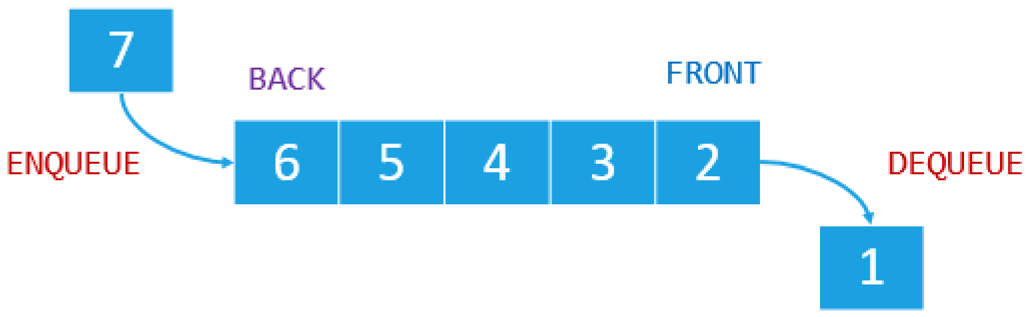
```
person = {name: 'bob', height: '6 ft', weight: '160 lbs', hair: 'brown'}

person.each do |key, value|
  puts "Bob's #{key} is #{value}"
end
```

```
Bob's name is bob
Bob's height is 6 ft
Bob's weight is 160 lbs
Bob's hair is brown
```

A person's attributes being stored in a hash and then output using iteration
From Launch School (Launch School, n.d.)

The key-value pairings allow data to be stored in a dictionary-like format, where you can search the hash by key, and find the associated value (like looking up a word in the dictionary and finding the associated definition). A phonebook could also be considered a hash structure, where each person's name is the key, and their phone number the connected value (Castello, 2019). Compared to arrays, hashes are best used when data needs to be associated with a label, and need to necessarily be in an ordered list. Rather than declaring and using multiple variables, all of that data could be stored in a hash instead, as seen in the example above. And to be even more efficient, arrays can also be used as a value in a hash, allowing a single key to hold multiple values (Launch School, n.d.).

QUEUES
Queues are a data structure that function similarly to their real world counterparts. Like arrays, they are ordered lists of elements, however the elements can only be inserted at the end of the queue and removed from the front, as seen in the diagram below.

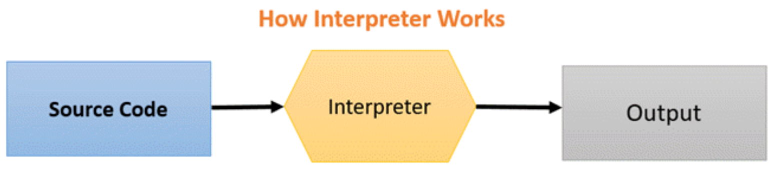


A Queue Data Structure
From JavaScript Tutorial (JavaScript Tutorial, n.d.)

This is known as the First-in, First-out principle, and is what makes the Queue different from other types of data structures. Queues in Ruby are useful as they can serve as simple implementations of waitlists. For example, if you were to develop an application that served customers, a queue could keep track of which customers to serve first, based on a first come, first served basis. Queues are also often used to manage shared resources such as printers (Castello, 2019). In Ruby, a queue's size can also be measured, or limited, to prevent services from being overburdened.

QUESTION 5

When writing computer code, we generally use a high-level language, one that is readable and understandable by humans. This is source code, and must be converted to machine code before it can actually be read by a computer. There are two ways to achieve this, by using either an Interpreter or Compiler (Programiz, n.d.).

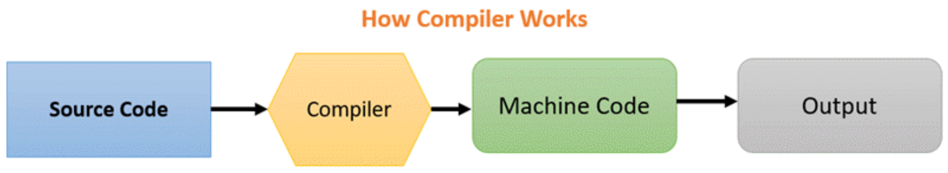


How Interpreters Work
From Guru99.com (Smith, 2022)

INTERPRETERS

As seen in the previous diagram, the source code is read by the interpreter and executed immediately, line by line. Therefore, no machine code is actually generated, and instructions are read directly from the source code. This allows interpreted code to be flexible and used across different computer types (such as Windows and Mac computers), as the source code is sent directly to the computer, where it is then read and executed. Javascript is an example of an interpreted language that is extremely common in today's world. Often when browsing websites, Javascript will be sent to the user's computer over the internet, and executed locally on their machine, which enables certain functionality and even design features on websites (Coding Mentors, 2018).

The line by line nature of Interpreters allows for easier debugging and is more friendly for beginner programmers; if a line of code produces an error when executed, it can immediately be fixed before any other code is interpreted. However there are also drawbacks to the line by line code execution. For one, the process is generally slower than a compiled piece of code, and having to read directly from the source code means the code has to be shared with the public if they are to run the code (Programiz, n.d.).



How Compilers Work
From Guru99.com (Smith, 2022)

COMPILERS

Unlike Interpreters, a Compiler will read all the source code at once, and convert it to Machine Code, which is only processable by the computer. This process, while a little slower than reading single lines of code like an interpreter, only has to be done once, and ultimately results in a faster and more efficient program once compiled (Programiz, n.d.). Due to the entire source code being compiled at once, it is a little more difficult to debug than interpreted code, as programmers have to sift through their source code to find the problematic lines.

There is also more code privacy with compiled code, as the source code does not need to be directly interpreted by an end user's computer. Though this means that compiled code is not as flexible as interpreted code, as it cannot be as easily read by different computer platforms (Coding Mentors, 2018). Some compiled languages are more portable, as their source code is first converted into Object Code. Object code is an intermediary code which is not immediately readable by the computer processor, but is converted into machine code at run time. This method of conversion exploits the advantages of both compilers and interpreters (Smith, 2022).

QUESTION 6

JAVASCRIPT

Considered a core technology of the current day World Wide Web, JavaScript is the most popular programming language in the world (Veeraraghavan, 2022). JavaScript was created in 1995, with the purpose of making web pages more dynamic, user-friendly and responsive to user actions. Even today, JavaScript is what enables the majority of webpages to be interactive. Its popularity is a major boon for JavaScript, as large online communities provide an abundance of invaluable resources for new developers learning the language, as well as extended functionalities through the use of third party addons. JavaScript also uses coding principles such as object-oriented, functional and imperative programming (Simplilearn, 2021).

JavaScript is an interpreted language, which brings even more benefits. For one, there is no need to compile the code, as it is run directly in the user's browser, resulting in speedy performance. Being primarily client-side also reduces loads on servers, with simple applications not needing servers at all. Coupled with its relatively simple syntax compared to other languages such as C++, JavaScript is a good choice for beginner programmers (freeCodeCamp, 2019). JavaScript is also highly versatile: if you wished to focus on front-end development, Angular in conjunction with JavaScript would make it possible. And Node.js would provide the same for back-end development. Electron and React also allow the development of desktop, mobile and web applications, all with the use of JavaScript.

Though JavaScript has plenty of advantages, as evidenced previously and with its popularity, there are also some drawbacks. As JavaScript is executed on the client side, the source code is visible to the public, and any bugs or oversights could be exploited with malicious intent. With the lack of confidence in security, some users have resorted to disabling JavaScript entirely within their browsers (freeCodeCamp, 2019). The code being executed on client machines also results in different outputs based on the browsers being used, as each browser may interpret the code differently. This increases development time as programmers have to test their code in all major browsers (freeCodeCamp, 2019).

PYTHON

Another highly popular programming language in the current world is Python. The language has its beginnings in the late 1980's and early 1990's, designed to be an easy-to-learn language with simple syntax. In fact, Python's English-like syntax and beginner-friendly nature is one of its major benefits, allowing newcomers to quickly grasp programming fundamentals, concepts and practices (Altexsoft, 2021). The language is also general-purpose, allowing it to be used in a wide range of applications and fields such as; web development, data analysis, software prototyping, system administration and much more. Similar to other popular languages, Python supports object-oriented programming, allowing for more complex applications to be built. Dynamic semantics is also supported, meaning that variable type need not be declared, as it is automatically recognized on code execution (Altexsoft, 2021).

With all of its features and simplicity, Python not quickens the learning process, but also speeds up software development. It also comes with an expansive standard library, and access to numerous additional libraries, which can streamline work in various fields such as mobile app development, game development, machine learning, data manipulation, web services and more (Basel, 2018). With access to these libraries and other frameworks, making programs in Python will likely take fewer lines of code compared to other, more verbose languages such as C++ (Altexsoft, 2021). Python also compiles its code to an intermediary bytecode, before being interpreted by Pythons Virtual Machine (PVM). This allows Python to be highly portable and versatile, being usable across Windows, Linux or

Mac systems. Python can also easily be used in conjunction with other languages such as C, Java and .Net, borrowing functionality that Python itself may lack (Altexsoft, 2021).

While Python has its many advantages, there are also some drawbacks. For one, being an interpreted language means there are speed limitations, especially compared to other popular, compiled languages. Python also uses a mechanism called Global Interpreter Lock (GIL), which only allows one sequence of code to be executed at a time. This ensures thread safety and increases performance on single-threaded CPUs, though performance on modern, multi-threaded processors suffers (Basel, 2018). Objects in Python can use up more memory than needed, and has generally poor memory management even with it's garbage collector. Overly high memory consumption makes Python a poor choice for memory intensive tasks (Basel, 2018). This flows into Python's mobile and front-end development issues, as the excessive memory and energy consumption are far too much for mobile hardware to afford. Coupled with the fact that Android and iOS do not natively support Python, and the availability of other more popular mobile-development platforms, makes Python a second-rate choice for smartphone app development (Altexsoft, 2021).

QUESTION 7

Computing technology is so integrated in today's society, that IT professionals and developers are more involved in the average person's life than ever before. Computers and software are used to store vital information, control critical infrastructure such as electrical grids and transport networks, and even make decisions such as whether to approve bank loans or offer someone a job (Pancake, 2018). Therefore, it is imperative that IT professionals are aware of their work's effects on people's lives, and the importance of serving society through ethical conduct.

PERSONAL INFORMATION

Consider what online and computer based services are used in an average person's day to day life. Shopping, finances, medical appointments, education, social and professional networking and the news a person consumes is all interwoven with the digital world. And all of the personal information and data regarding these services are shared with the businesses that operate them. The typical consumer has all of their personal information accessible, unfiltered by a number of enterprises they might loosely be involved with (Ritter, 2020).

This practice of collecting personal data is quite common and almost considered a part of normal business operation in today's world. Companies state their intentions are to use the data they collect to gain a better understanding of their customers, so that they can improve their services and encourage customer loyalty, and repeat business. And while initially this seems well intentioned, there are numerous ethical concerns for IT professionals, regarding not only data collection, but also the handling and security of the data (Ritter, 2020). Data that is collected on its own could seem harmless at first, however the manner in which it is used could be unethical.

For example, when applying for a loan or credit, banks or financial institutions would traditionally look at a person's income level or employment history, to make informed decisions on whether to approve the credit. If a human were to make this decision, they could tell the borrower the exact reason they were approved or denied credit. However in a digital world, these decisions could be made by machine learning algorithms, using any information gathered on the burrower. There could be scenarios where borrowers are denied a loan based on their postcode, if an algorithm determines others in the same area code have bad credit. And if an applicant's postcode is being considered, are other factors such as their race or religion also being looked at? In this case, the developers of the algorithm are responsible for ensuring that any decisions are made using only legitimate criteria and data; and that the system is not being unjustly discriminatory (Datta, 2017).

Different government bodies have enacted laws and regulations to allow equal opportunities for all, such as the *Equal Credit Opportunity Act* in the USA, or the *NCCP Act 2009* in Australia (ASIC, n.d.). These legislations state that all are to be given equal opportunities when it comes to money lending. Alongside this, borrowers are owed an explanation if their application is denied (Datta, 2017). So if a company is using computer algorithms to determine whether loans should be approved or denied, the developers must ensure that all applicants are given a fair and equal opportunity, and in the case of denial, it should be possible to provide the applicant with the reasons for it. Legislations that have been enacted more recently are the *European Union's General Data Protection Regulation* (GDPR) and the *California Consumer Protection Act* (CCPA). These legislations encourage companies to adopt more ethical data practices, as any data that they have collected on a customer must be disclosed upon said customer's request. Not only this, but said data must be erased if demanded by the customer (Ritter, 2020). This increased transparency results in less unethical data collection and misuse. A proactive approach would also ask for a customer's consent before collecting their personal information, and notify them how it is being used. And for safety, the data could be deleted after a certain period of time, without the need for customers to request it specifically (Ritter, 2020). Such practices if integrated would be considered ethical handling of the consumer’s data.

GPS TRACKING

The Global Positioning System (GPS) has been increasingly adopted over the past 20 years by private and public enterprises to provide location based services and tracking. With most people likely owning multiple smart devices capable of GPS tracking, the ethical handling of location data is more critical now than ever (Michael et al., 2006). The ever increasing reliance on smart devices in the modern world means that your location is likely being tracked at any given moment. And while this data could be used to improve services and provide convenience, ethical issues are raised when this data is used as a form of surveillance and control (Michael et al., 2006, #).

GPS tracking could be considered an intrusive method of supervision, and has been used as such by businesses to monitor employees, law enforcement to track parolees and suspected terrorists, and even parents to keep track of their children. Though the controlling bodies may have good intentions when they choose to track individuals, there are ways the tracking data could be misused to the detriment of the individual. For example, with the case of tracking parolees, it could be argued that surveillance will prevent further crimes from occurring. However, on a parolee who is unlikely to offend again, imposing restrictions could inhibit their rehabilitation (Michael et al., 2006, #).

The greatest ethical concerns come from the information that can be deduced from analysing an individual’s movements. For example, while keeping track of children for their safety may seem well intentioned; it can also be a great invasion of their privacy, especially when considering that children likely have no choice in being tracked. Similarly, businesses may track their employees to increase efficiency. However they are just as likely to monitor employee movements outside of work hours, and use this information to unfairly discipline them (Michael et al., 2006, #). An IT professional is responsible for ensuring that only relevant data is used, without breaching the tracked person’s privacy. For example, data on employees outside of work hours would be considered irrelevant, and should not be tracked. Likewise, if a parent was to keep track of their child, they would only be given the relevant information at the necessary times; such as a time when their child does not show up to school.

Laws and legislature that affect GPS tracking differ between countries and even states. In Australia, each state has different rules with regards to tracking employees for example, however many guidelines refer to *Surveillance Devices Acts* and *Invasion of Privacy Acts* (Trackometer, 2021). While GPS tracking is legal, to ensure ethical use, tracking should only be done with the consent of the individual being tracked. The individual should also be given plenty of notice before tracking begins, and they should be aware of how and why the data will be used. The safeguarding of the data is also of utmost importance, and is the responsibility of the IT professional. Efforts should also be made to not track the individual outside of relevant locations and times, to provide them with as much privacy as possible (Watts, 2021).

UBER GREYBALL

Uber is a ride sharing service that was launched in 2012 as an alternative to traditional taxi cabs. The service has since grown to be usable in over 700 cities around the world, continues to expand and has spawned other, rivalling ride sharing platforms (Everette, 2021). Unlike traditional taxi services, Ubers do not need special licences, and are driven by anyone using their own personal cars. Uber boasts cheaper fares, quicker service and more convenience compared to other taxi services, and has at times caused friction with said taxi services whenever Uber has been introduced to a new location.

Ride sharing services in their infancy have existed in legal and regulatory grey areas. Uber was often introduced into cities without seeking permission from local authorities. The usage of noncommercial drivers with private vehicles also concerned authorities regarding the safety of the service. Uber operating without authorisation, along with local taxi services becoming aggravated with lost business, resulted in certain parties looking to clamp down on the ride sharing service. Though agreements were eventually reached, and legal-frameworks developed, Uber was disadvantaged with the increased legal costs and fines its drivers would receive. And so Uber Greyball was developed, to help Uber avoid law enforcement and regulation. (Isaac, 2017).

Greyball would use data collected from the Uber app, among other techniques, to identify authorities and regulators and circumvent them. Authorities would conduct sting operations, where they would use the app to book a ride, and once the driver arrived, they would receive a fine or have their vehicle impounded. However, using Greyball, the app would identify the customer as a potential government employee or regulator, and avoid sending a vehicle (Isaac, 2017). Uber would monitor phone activity near government offices, and any user accessing the app frequently would be marked as a likely regulator. The credit card used by an individual would also be looked at to determine if it were tied to police unions. Uber would even flag certain phone models, as they were often used by government officers that were trying to conduct sting operations. Social media accounts of individuals were also monitored, to see if they were linked to local law enforcement (Isaac, 2017). All of this personal information was being collected and used to mark individuals as potential authority figures which were to be avoided to continue providing an unregulated service. When Greyball was exposed and it was revealed that Uber was deliberately using such methods to avoid regulation and sidestep authorities; executives did not initially deny the claims and held firm that the system was designed to protect its drivers. It maintained that Greyball was just part of its larger, Violation of Terms of Service system, that would analyse credit card information, location data, device information and other factors to determine whether a customer was legitimate or acting in bad faith (Reuters, 2017).

The usage of Greyball can be considered a breach of ethics in both the manner in which it gathered personal information to flag individuals, and the deliberate skirting of the law and avoidance of regulation. The breaches occurred because it was considered more profitable for Uber if they could operate outside of government regulations. However, for the safety of its customers, Uber should have consulted and worked with local authorities to develop rules and regulations beneficial for both parties. Certain regulations are present for the safety of the public, and using noncommercial drivers with private vehicles could be considered putting riders at risk, even if Uber had their own process of examining drivers. Again, Uber could have worked with regulators to highlight their hiring practices, and avoid any ethical breaches. The willingness shown by Uber to avoid adhering to rules raises concerns; what other rules are they skirting, just to increase profitability? If the data they collect can be used to determine if an individual is a government official, could they also determine how much expendable income a user has? Using personal information, it could be possible for Uber to charge each user a different fare, based on assumptions on what that user is willing to pay. This would be another major ethics breach, and highlights how important ethical data practices are. In this case, much of the information that Uber was scraping for its Greyball system would be considered irrelevant for a typical customer, and should not have been used to avoid regulations.

QUESTION 8

In programming, control flow is the order in which code is executed from a script, typically from the first line to the last. Ruby, like other programming languages, has structures known as conditionals and loops, that can alter the control flow (MDN Contributors, 2022). Control flow is apparent in many applications, for example, in a video game, the application would run scripts depending on the player's inputs: such as calling a 'jump' function if the player presses the jump button. The player character's life points would also be tracked, and a 'game over' script would be executed on the condition that the life points have reached zero.

IF, ELSE, ELSIF

In Ruby, one of the most commonly used conditional statements is the **'if'** statement. Simply put, it is used to check whether an expression is true, and allows for code to be executed if so. This can be branched using the **'else'** statement, to execute other code if the expression returns false. 'Else' and 'if' statements can also be combined into **'elsif'**, which allows the creation of multiple branches (Bodnar, 2021). The usage of these statements can be seen below:

```
age = 17

if age >= 18
     puts "You can enter the club"
elsif age < 0
     puts "Please enter a valid age"
else
     puts "You are underage. 18+ only"
end

# output => "You are underage. 18+ only"
```

The **'AND'** and **'OR'** operators are also usable alongside comparison operators in 'if' statements to check for even more conditions. All together, these statements and operators allow for greater management of the program's control flow.

CASE

Similar to the 'if else' statement is the **'case'** conditional statement, which is a type of selection control flow. With 'case' it is possible to create multiple branches in a simpler way compared to using combined 'if' and 'elsif' statements (Bodnar, 2021). It works by comparing an input value with a list of values; if a match is found its respective code is executed, **'else'** other code is run.

```
weather = "rainy"

case weather
when "sunny"
    puts "Put some sunscreen on."
when "rainy"
    puts "Take an umbrella."
when "storm"
    puts "Please stay home."
else
```

```ruby
    puts "Sorry, cannot read the weather."
end

# output => "Take an umbrella."
```

WHILE, UNTIL
Unlike the previous conditional statements, **'while'** and **'until'** are control flow statements that allow repetitive execution of code based on a boolean condition, or simply a 'loop'. A 'while' loop will execute the code enclosed as long as its condition returns true, whereas an 'until' loop will execute while the condition is false.

```ruby
# This loop will output the cycle number 10 times

i = 0           # Initialisation of the loop
while i < 10    # Testing condition
   i = i + 1    # Updating the counter and executing the following statement
   puts "This is cycle number: #{i}"
end
```

These loops have 3 important parts, the **Initialisation**, **Testing** and **Updating** (Bodnar, 2021). As seen above, the 'initialisation' is where the loop counter will be declared. Following this is the 'testing' condition; statements in this block will be executed until the testing returns false (in a while loop). The final part is the 'updating', where the counter is incremented. Programmers need to take care with the update stage as improper handling can lead to endless loop cycles.

FOR, EACH
The previous 'while' and 'until' statements will keep looping until a certain condition is met, however if you wish to loop a known number of times, **'for'** loops are to be used. Specifically, a 'for' loop works within a declared range, and a block of code is executed for each element within that range (Bodnar, 2021). This makes the for loop perfect for iterating through arrays, as an array's length can be measured and used to initialise the loop. The **'each'** method can also be used on an array to iterate through each element, similar to a for loop. When writing an 'each' statement, the element is put between pipes and serves as a placeholder for the object pulled from the current iteration of the array. How 'for' and 'each' loops are used can be seen below:

```ruby
# An array of planets in the solar system
planets = ["Mercury", "Venus", "Earth", "Mars",
"Jupiter", "Saturn", "Uranus", "Neptune"]

# Printing out all planet names using a for loop
for i in 0...planets.length
    puts planets[i]
end
# Printing out all planet names using the each method
planets.each do |planet|
    puts planet
end
```

For and Each loops in Ruby
(Bodnar, 2021)

BREAK, NEXT

Conditional statements are helpful in controlling iterations of loops, but for even more control the **'break'** and **'next'** statements can be used. 'Break' is frequently used within 'while', 'for' or 'case' statements; when a computer reads 'break', it will skip all following code within the block and exit the loop. Likewise, when the computer reads a 'next' statement within a 'while' or 'for' loop, it will skip to the next iteration of the loop. Usage of these statements can be seen below:

```ruby
# An endless while loop
while true
    # Generating a random number between 1 and 30, then printing it.
    r = 1 + rand(30)
    print "#{r} "

    # The loop only ends if the random number is 22
    if r == 22
        break
    end
end
```

Using break to exit a while loop
(Bodnar, 2021)

```ruby
# A loop that increments from 1 to 100
num = 0
while num < 100
    num += 1

    # If the number is even, the loop skips to the next iteration
    if (num % 2 == 0)
        next
    end

    # The number is only printed if it is odd
    print "#{num} "
end
```

Using next to skip every second iteration
(Bodnar, 2021)

QUESTION 9

Type Conversion and Type Coercion are used in programming languages as a way to change data from one type to another; such as changing a string to a number. Type Conversion, as the name implies, converts the value either implicitly or explicitly, whereas Type Coercion changes values implicitly (MDN Contributors, 2021).

```
:foo.to_s          # => "foo"
10.0.to_i          # => 10
"10".to_i          # => 10
"Foo10".to_i       # => 0
[1, 2, 3].to_s     # => "[1, 2, 3]"
Object.to_s        # => "Object"
```

Explicit Casting in Ruby
(de Bruijn, 2018)

In Ruby, explicit casting methods are used to convert data easily from one value to another. When used, these casts always return a value in the requested data type, even if the data may not be what is expected. As seen in the example above, **#to_s** always returns a string, and **#to_i** always returns an int, no matter the initial input. Forcefully converting data values in this manner can cause errors or loss of data, which is where implicit type coercion comes into play (de Bruijn, 2018).

```
10.to_int                 # => 10
10.0.to_int               # => 10
name = "world!"
"Hello " + name           # => “Hello world!”

# Unsuccessful coercions
"10".to_int               # => NoMethodError
"foo10".to_int            # => NoMethodError
[1, 2, 3].to_str          # => NoMethodError
{ :foo => :bar }.to_str # => NoMethodError
{ :foo => :bar }.to_ary # => NoMethodError
Object.to_str             # => NoMethodError
```

Implicit Coercion in Ruby
(de Bruijn, 2018)

Type Coercion is different from conversion in that a value is only returned if the original type acts like the desired value. For example, **#to_str** will only return a string if the original type can act like a string, otherwise an error will be raised. This strictness allows us to be certain that a variable will behave the way we expect it to (de Bruijn, 2018). Methods such as **#to_str**, **#to_int**, **#to_ary** are all examples of implicit coercion methods, but they are not the only way Ruby uses type coercion. Implicit coercions are also used when objects are combined using the **+** operator.

There are also other, more resilient coercion methods used in Ruby: **String(...)**, **Integer(...)**, **Float(...)** and so on. These methods first try an implicit conversion (#to_str), and if that fails, an attempt is made at an explicit conversion (#to_s). This increases the chances that a user gets the value they are looking for and reduces the risk of errors and data loss (de Bruijn, 2018).

QUESTION 10

In programming, data types are used to classify data so that the computer knows how it can be handled, and what operations can be performed. In the real world, humans can perform maths operations on different types of numbers such as whole numbers, integers and even irrational numbers. However, if someone were asked to solve an equation such as **'hello' + 45**, they would be perplexed. This is because **'hello'** is a word and **45** is a number, two different data types that don't really interact with each other (Hogan, 2017). Computers are very much the same as humans, and can only manipulate data types in certain ways, therefore it is imperative that the correct data types are used so that you don't receive an unexpected result.

With some programming languages, the data type of a variable needs to be known at compile time, this is known as statically-typed. Ruby on the other hand, is dynamically-typed, where data types need not be explicitly declared. Instead, the interpreter will determine the data type at runtime (Hogan, 2017). This also allows variables to hold different data types, and convert between data types without needing to store data in other variables. Also, being an object-oriented language, Ruby's data types are all implemented as classes. Therefore every Integer, String, Float, or other data type is actually a Ruby object; and all contain a **class** method which identifies the data type of itself.

```
42.class                    # Integer
(42.2).class                # Float
["Sammy", "Shark"].class    # Array
true.class                  # TrueClass
nil.class                   # NilClass
```

Using .class to identify data types in Ruby
(Hogan, 2017)

INTEGER
Integers or **int** in Ruby are whole numbers that can be positive, negative or even 0. Integers can have mathematical operations performed on them, though if all values in the equation are integers, the result will also be an integer. As seen in the example below, when dividing two integers would normally result in a decimal number, instead a rounded whole number is returned.

```
# Maths operations using integers
5 + 7      #  => 12
2 * 8      #  => 16
5 / 2      #  => 2
```

FLOATING-POINT NUMBERS
Otherwise known as **float,** floating-point numbers are numbers that contain a fractional part, or decimal. Ruby considers any number written without decimals as an integer, and any with a float. Performing operations with a float involved will return a float, even if integers are involved.

```
# Maths operations using floats
5 + 2.7    #  => 7.7
2 * 8.5    #  => 17.0
5.0 / 2    #  => 2.5
```

BOOLEAN
Boolean, also known as **bool**, represents truth values and is used to solve logical problems. In Ruby, Booleans are represented as either **true** or **false**. Many mathematical comparison operations will result in either a true or false value, which can then be used to execute other pieces of code or functions.

```ruby
# Maths operations returning Boolean values
5 > 100                 #  => false
7 < 12                  #  => true
12 == 4                 #  => false
12 == 6 + 6             #  => true
"Hello" == "world!"     #  => false
"this" == "this"        #  => true
```

STRINGS
A sequence of one or more characters is known as a string. The characters may be letters, numbers or symbols, and in Ruby are typically enclosed within single or double quotes. Strings are essential in communicating information to users, and receiving information from the user. In Ruby, some mathematical operations can also be performed on strings, as seen below.

```ruby
# A string
"This is a string."

# Also a string
"53"

# Maths operations using strings
"Hello " + "World!"     #  => "Hello World!"
"hello" * 3             #  => "hellohellohello"
"hello" + 5             #  => TypeError
"hello" / 2             #  => NoMethodError
```

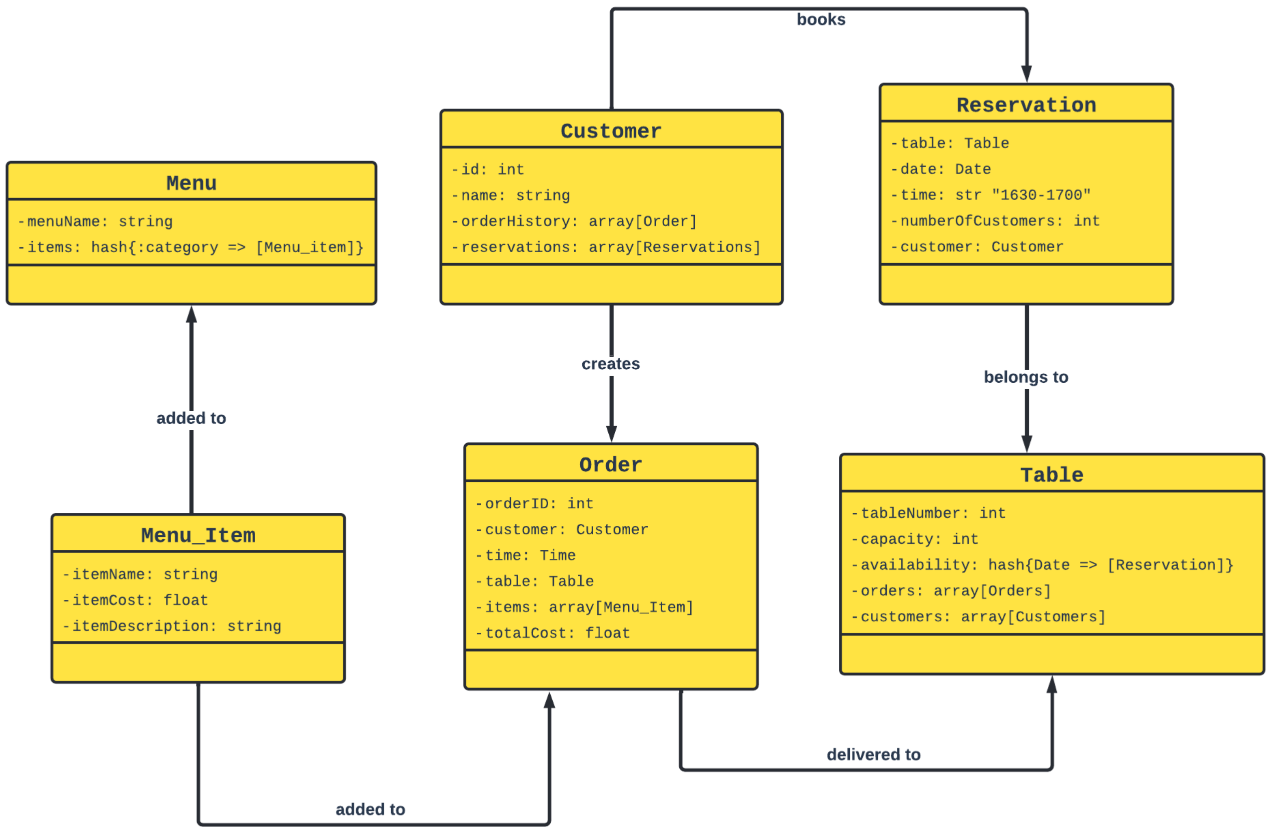
SYMBOLS
Considered a special data type, symbols act as labels or identifiers in Ruby. While strings are generally intended to be worked on and manipulated, symbols are immutable and cannot be changed. Each string in Ruby is an object and will have its own unique memory location, even if two string values are identical. A symbol however, will always refer to the same memory location. This makes symbols perfect to use as keys in hashes, and results in less memory usage and better performance (Hogan, 2017).

```ruby
# Symbols in Ruby
:first_name
:last_name
:student_id
```

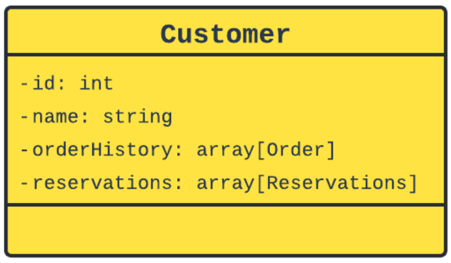
Arrays and Hashes are also considered data types in Ruby, and can also be stored as variables. Though they are also considered data structures as their main purpose is to store data. More information on arrays and hashes can be found in the previous, data structures section.

QUESTION 11

For this problem, we assume the interactions and functions generally present in a restaurant setting will be handled by an application. From management of table reservations, to menu browsing and even ordering. The classes are intended to store data in an organised manner, and allow functions to be performed using said data. A general layout of the classes used in such an application can be seen in the diagram below:



Simple UML Class Diagram of a restaurant application

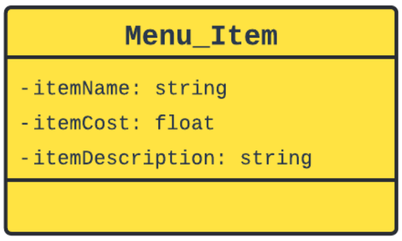

Customer
-id: int
-name: string
-orderHistory: array[Order]
-reservations: array[Reservations]

CUSTOMER

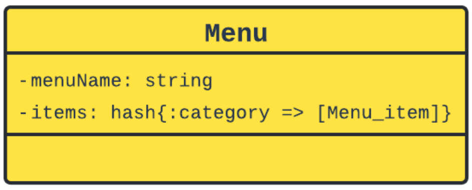
The restaurant is likely to have many customers visiting at any given time, and it is important to keep track of them in the application. Customers should be able to reserve a table, make orders and pay through the application. Each customer should have a **unique ID**, a **name**, and an **order history** in case they wish to dispute a previous order, or if the restaurant wishes to implement a loyalty program. Also any **reservations** the customer has made will be stored in an array, which can be accessed if the customer wishes to cancel or make changes.

MENU_ITEM

A restaurant's primary purpose is to serve food and drinks, and so for this application, each menu item is to be an object. The item's **name**, **cost** and a **description** will be stored as attributes, and can be changed by the restaurant if they so wish (in the case of updating recipes, ingredients and costs) Menu items are able to be displayed in multiple menus and added to multiple orders by customers.



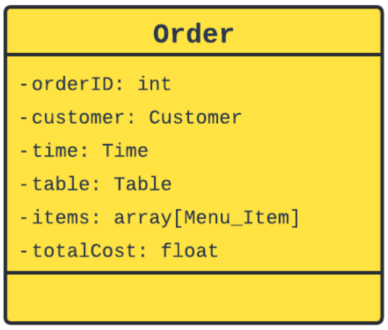
MENU



This application will allow for multiple menus; they could be seasonal, lunch or dinner, cocktail or kids menus. Each menu will have a **name** to help identify it, along with an **array of menu items** keyed with a **category** and stored in a hash. A lunch menu for example, may have **:coffee**, **:sandwiches**, **:pizza**, **:drinks** as keys in the Menu hash, and each key having an array of **Menu_items** assigned as a value. This method of organising data will streamline the menu creation process and make ordering easier for customers.

ORDER

Customers will be able to create an order through the app, simply by browsing the Menu and adding menu items to their order. Each order will have a unique **Order ID**, the **Customer** placing the order, the **Table** the order will go to, an array of **Menu items** to be delivered to the table, **time** of the order and the **total cost**. The cost can be determined by adding the costs of all menu items; and allows the customer to pay directly through the app. By saving the time of order, the app can keep track of how long customers have been waiting for their order to be fulfilled, and the restaurant can prioritise orders in this manner.



TABLE

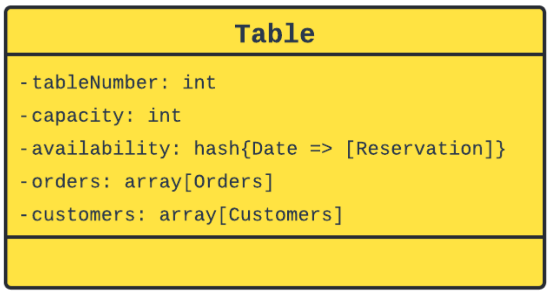


Table
-tableNumber: int
-capacity: int
-availability: hash{Date => [Reservation]}
-orders: array[Orders]
-customers: array[Customers]

The restaurant, like any other, is assumed to have numerous tables that can seat customers. It is important to keep track of which tables have customers seated, have orders pending and are free to be seated or reserved by new customers. Therefore each Table should have an identifying **tableNumber**, the **capacity** of customers it can seat, a hash of Reservations on each day to indicate **availability**, a list of **orders** currently pending, and the **customers** seated at the table, if any.

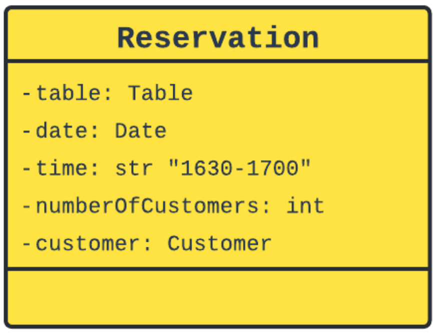
Each Table will have an array of time slots represented by Reservation objects. Each Reservation object has a customer attribute, to indicate if a customer has a booking with the current Table. If the customer attribute is empty, the table is available for the respective time slot. Functions could iterate through each time slot, for each table to find all available times. This can be displayed to the customer to make booking a table quick and easy.

```
# The function to display available bookings on a selected date


Look at each Table in the restaurant
   Look at each Reservation time slot for the Table on the selected date


      Does this Reservation time slot have a Customer already?
         Yes - The Table is already booked - GO TO NEXT RESERVATION
         No - The Table is available at this time
```

```
        Is there a Customer booked within the next 2 hours?
           Yes - The Table is not available for long enough - NEXT RESERVATION
           No - ADD TO AVAILABLE TIMES

Display AVAILABLE TIMES after looking at each Reservation time slot for each Table
on the selected date.
```

Pseudocode of the process of displaying available reservation times

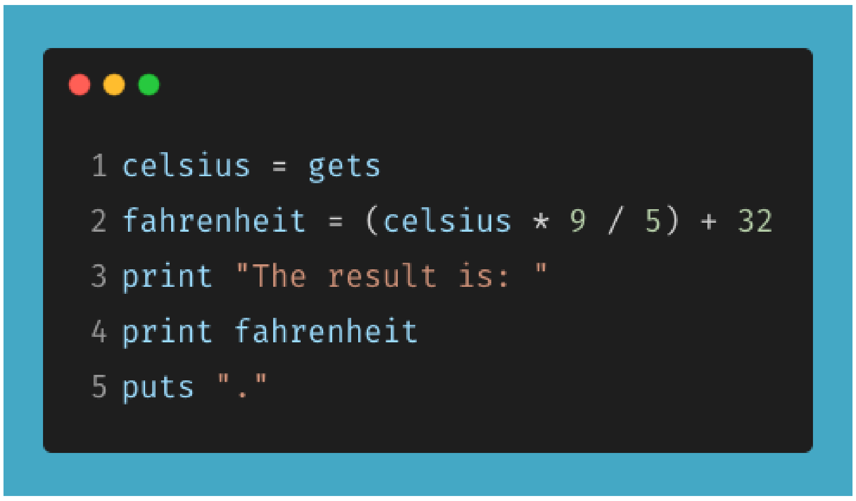
RESERVATION



Customers should be able see when tables are available, and have the ability to book a table at a specific time. The Reservation class represents a time slot belonging to a Table that can be booked by a customer. Therefore the class needs to track the **Table** that is reserved, **Date** and **Time** of reservation, **number of customers**, and the **customer** that has made the reservation. Even if a customer walks in without a reservation, and sits at a free table; this can be logged in the application as a reservation from the time the customer sits down, just to indicate that the table is no longer available.

When a customer reserves a Table, the respective Reservation object and following 3 objects will be updated with the Customer information, resulting in those time slots being marked as booked. We book 4 time slots at a time, as each reservation object represents a 30 minute slot of time *(time = “1630-1700”)*; this gives customers a very generous 2 hours of time. If the customers leave early, any excess reservation objects will have their Customer attribute cleared, to indicate the time slot is once again available.

QUESTION 12



```ruby
celsius = gets
fahrenheit = (celsius * 9 / 5) + 32
print "The result is: "
print fahrenheit
puts "."
```

Original code snippet

The provided code snippet looks to be a simple program that takes user input of temperature in celsius, converts it to fahrenheit and prints the result. However the program does not run correctly, as when a user inputs a valid number, a **NoMethodError** is returned. This is because the celsius variable is saving the user input as a string. If a user were to input '25', then the computer would attempt to solve the following equation.

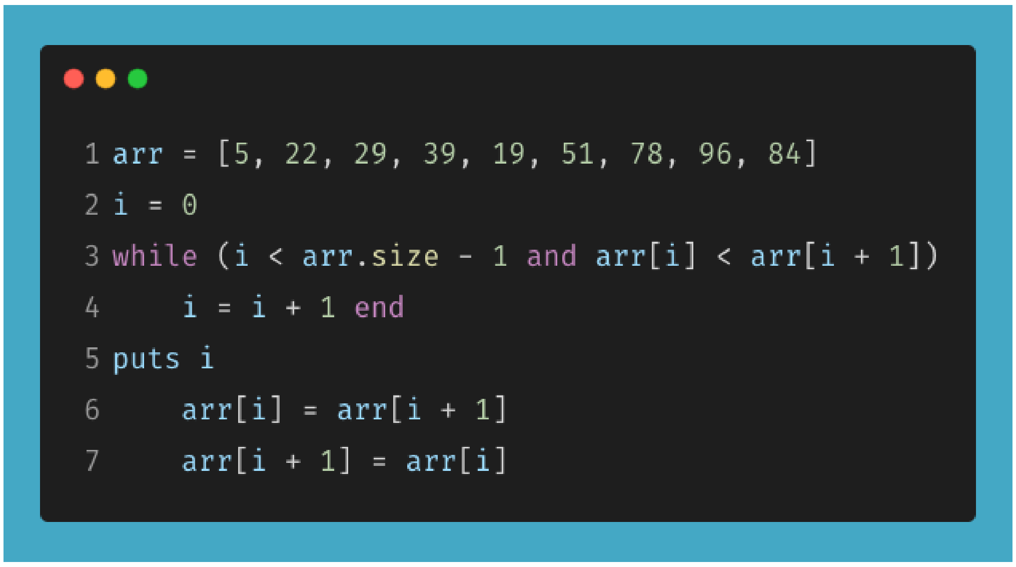
```ruby
fahrenheit = ("25" * 9 / 5) + 32
```

While the computer does know how to multiply the string with an integer, it does not know how to divide, and throws an error. To remedy this, first we use the **chomp** method on 'gets' to remove any extra, special characters. And then use the **to_f** method to convert and store the user input as a float. We could also store the value as an **integer** which would provide a nice rounded, whole number, but for accuracy it is better to stick with **float**. This corrects the issue the computer has with trying to solve the equation. To take an extra precautionary measure, we can also print a message to notify users the program's function, and what they are expected to input. This can help avoid users from inputting any characters that may cause errors.

```ruby
puts "The Celsius to Fahrenheit Converter!"
puts "------------------------------------"
print "Please enter a value in CELSIUS: "
celsius = gets.chomp.to_f  #Convert and store the user input as float
fahrenheit = (celsius * 9 / 5) + 32
print "#{celsius}C is: "
print fahrenheit
puts "F."
```

The amended code

QUESTION 13



```
1 arr = [5, 22, 29, 39, 19, 51, 78, 96, 84]
2 i = 0
3 while (i < arr.size - 1 and arr[i] < arr[i + 1])
4     i = i + 1 end
5 puts i
6     arr[i] = arr[i + 1]
7     arr[i + 1] = arr[i]
```

Original code snippet

This program involves an array of numbers that get iterated through with a while loop, until it finds an element that is greater than the following element. At this point, the program ends the loop and attempts to swap the two elements, which is where an issue occurs. **Line 5** outputs index '3', which shows that the program is correctly identifying the problem element in the array. Though if we were to print the array after the elements have supposedly been swapped, we would see the following:

```
[5, 22, 29, 19, 19, 51, 78, 96, 84]
```

It can be seen that rather than swapping the numbers '39' and '19', the number '19' has just been duplicated. This is because **arr[i]** is assigned the value of 19, and then **arr[i + 1]** is assigned the value of **arr[i]** which as we know was just assigned 19. This results in the duplication, and the error in the program. To solve this, we can store the original arr[i] in another variable called **swap_number**, and retrieve it when assigning it to arr[i + 1]. The amended code can be seen below:

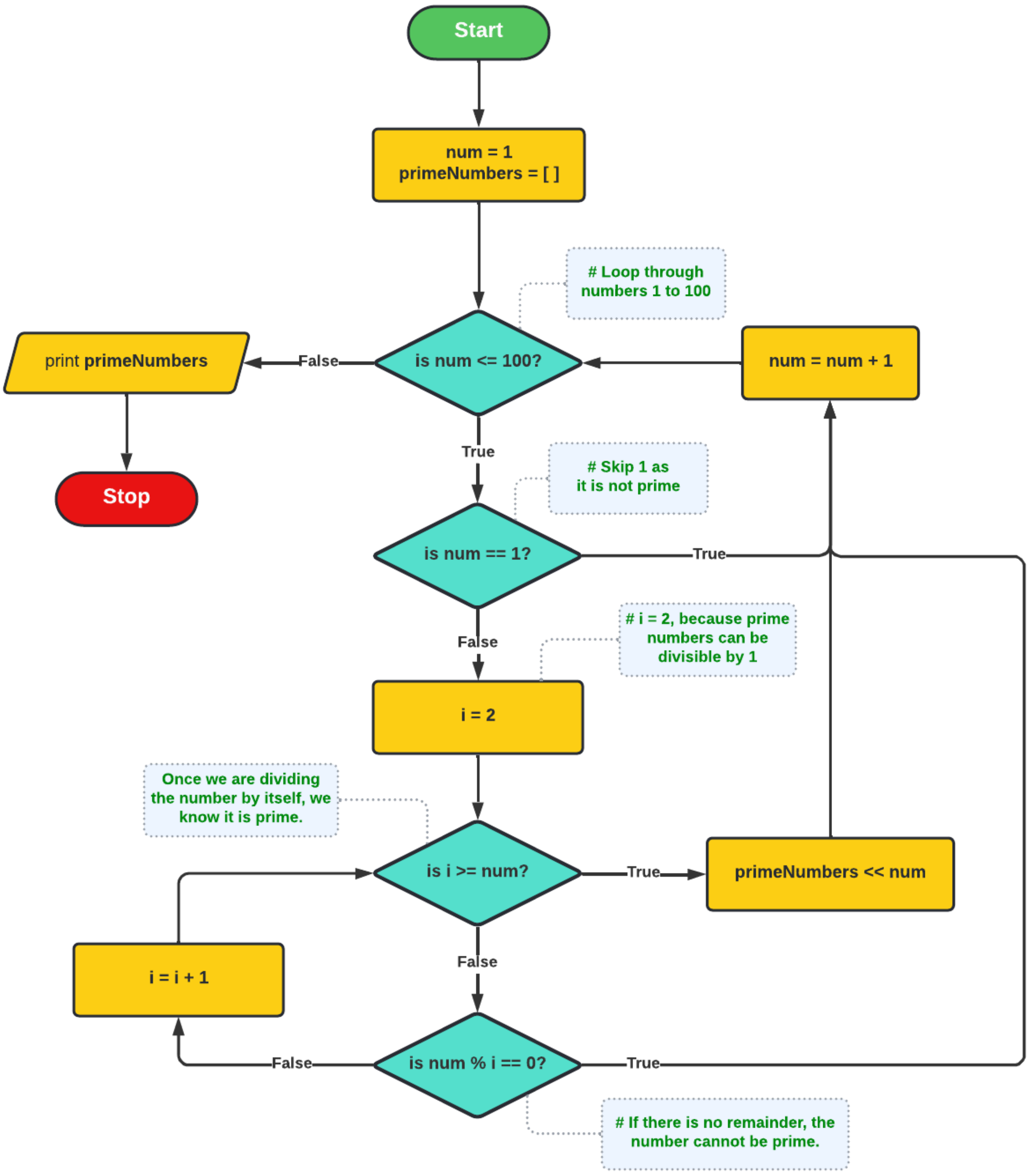
```
# The original array
arr = [5, 22, 29, 39, 19, 51, 78, 96, 84]

# This loop iterates through the array until it finds an element
# that is less than the following element. It then exits the loop.
i = 0
while (i < arr.size - 1 and arr[i] < arr[i + 1])
    i = i + 1
end

# We know that the element that needs to be swapped is at index 'i'
# And we know it must be swapped with the following number at 'i + 1'
# The number is stored in a variable while the elements are swapped
swap_number = arr[i]
arr[i] = arr[i + 1]
arr[i + 1] = swap_number
pp arr
```

QUESTION 14

To create a list of prime numbers from 1 to a 100, we first need to determine how to find a prime number. With an input number, we need to make the following decisions:

1. Is the number **whole** and **greater than 1**?
2. The number **cannot be divided** by any number other than **itself** and **1**?

If both are true, then the number is prime. For the first step, we can assure numbers are whole by using **integers**. And for the second step, we divide the number with every number before it, except for 1 and itself; if any division succeeds without a remainder, then the number is not prime. The process is repeated for every number between 1 and 100, as can be seen in the flowchart and following pseudocode:


Start
num = 1
primeNumbers = [ ]
# Loop through numbers 1 to 100
is num <= 100?
False
print primeNumbers
Stop
num = num + 1
True
# Skip 1 as it is not prime
is num == 1?
True
False
# i = 2, because prime numbers can be divisible by 1
i = 2
Once we are dividing the number by itself, we know it is prime.
is i >= num?
True
primeNumbers << num
False
i = i + 1
False
is num % i == 0?
True
# If there is no remainder, the number cannot be prime.

Flowchart of program that lists prime numbers from 1 to 100

```
For each number between 1 and 100

      If the number is 1 then
            GO TO NEXT NUMBER

      # Prime numbers can only be divisible by 1 and itself
      Set the denominator to 2

      Is the denominator equal to or greater than the number?
            Yes - Primes can divide by themself, therefore the number is prime
            ADD NUMBER TO LIST OF PRIMES AND GO TO NEXT NUMBER

      No - Divide the number by the denominator

      Is there a remainder?
            Yes - The number couldn't divide.
            INCREMENT THE DENOMINATOR BY 1

            No - The number divided by something other than 1 or itself.
            GO TO NEXT NUMBER

Once all numbers between 1 and 100 have been tested
PRINT THE LIST OF PRIMES
```

Pseudocode of the process of listing all prime numbers between 1 and 100

QUESTION 15

```
Is the temperature LESS THAN 15, AND it is raining?
      Yes - PRINT "It's wet and cold"

No - Is the temperature LESS THAN 15, AND NOT raining?
      Yes - PRINT "It's not raining but cold"

No - Is the temperature GREATER THAN OR EQUAL TO 15, AND NOT raining?
      Yes - PRINT "It's warm but not raining"
      No - PRINT "It's warm and raining"
```

Pseudocode to determine the weather

```ruby
# Get raining value from the user
raining = false
puts "Is it raining today? (y/n)"
if gets.chomp == "y"
    raining = true
end

# Get temperature value from the user
puts "What is the temperature today?"
temperature = gets.chomp.to_i

# Determine the weather
if raining && temperature < 15
    puts "It's wet and cold"
elsif !raining && temperature < 15
    puts "It's not raining but cold"
elsif !raining && temperature >= 15
    puts "It's warm but not raining"
else
    puts "It's warm and raining"
end
```

Ruby code to determine the weather based on user input

QUESTION 16

```ruby
# Clear the terminal
system 'clear'

# Initialise the skills data
skillsHash = {  :python => 1,
                :ruby => 2,
                :bash => 4,
                :git => 8,
                :html => 16,
                :tdd => 32,
                :css => 64,
                :javascript => 128}

# Empty hash which will track the user's skills
userSkills = {}

# Skill scores will be added to this total
codingSkillScore = 0

# Welcome message
puts "================ ACME Corporation ================"
puts "Welcome! Do you have any of the following skills?"

# Print out the list of skills ACME is looking for
skillsHash.each do |skill, value|
    puts " - #{skill}"
end

puts "--------------------------------------------------\n\n"

# Endless loop that will allow the user to enter multiple skills
# Will end if the user enters 'done'
while true

    print "Please enter a skill, or 'done' if you are done: "
    skill = gets.chomp.downcase.to_sym #turn the string into symbol so it can be searched for in
hashes

    # Terminate the loop if the user is done
    if skill == :done
        break
    end

    # Check to see if the input skill exists in the list of skills
    if skillsHash.key?(skill)

        # Check to see if the skill has already been recorded in the User's skills
        if userSkills.key?(skill)
            puts "\n*** We have already recorded #{skill.upcase}! ***"
        else
            # If not, add the skill to the User's skills list and add to the skill score.
            userSkills[skill] = skillsHash[skill]
            codingSkillScore += skillsHash[skill]
            puts "\n*** #{skill.upcase} recorded ***"
        end
    else
        # The skill is not recognised
        puts "\n*** Sorry, we don't recognise #{skill.upcase}! ***"
    end
```

```ruby
end

# Display the skill score when the user is done entering skills.
puts "\n==================================================="
puts "Your Coding Skill Score is: #{codingSkillScore}"
puts "==================================================="

# Set a threshold score which users must beat to be considered for employment
thresholdScore = 200

# If the users score falls below the threshold, they will be recommended skills to learn
if codingSkillScore < thresholdScore

    # Recommend what skills the user should learn.
    puts "\nTo improve your score, we recommend learning these skills:\n\n"

    # Search through each skill and see if the user knows it.
    skillsHash.each do |skill, value|
        if !userSkills.key?(skill)
            # if not known by the user, print it out along with its point value.
            puts " - #{skill} will increase your score by #{value} points."
        end
    end

else
    # User is skilled enough
    puts "\nYou have all the skills we need!\n\n"
end


# Closing message
puts "\n--------------------------------------------------"
puts "\t We hope to work with you soon!"
puts "================ ACME Corporation ================\n\n"
```

REFERENCES

Altexsoft. (2021, September 28). *Pros and Cons of Python Programming Language*. AltexSoft. Retrieved April 5, 2022, from https://www.altexsoft.com/blog/python-pros-and-cons/

Amazon AWS. (n.d.). *What is DNS? – Introduction to DNS - AWS*. Amazon AWS. Retrieved March 29, 2022, from https://aws.amazon.com/route53/what-is-dns/

ASIC. (n.d.). *Responsible lending*. ASIC. Retrieved April 10, 2022, from https://asic.gov.au/regulatory-resources/credit/responsible-lending/

Basel, K. (2018, June 20). *Python Pros and Cons (2021 Update)*. Netguru. Retrieved April 4, 2022, from https://www.netguru.com/blog/python-pros-and-cons

Bodnar, J. (2021, December 15). *Flow control in Ruby - control program flow in Ruby*. ZetCode. Retrieved April 6, 2022, from https://zetcode.com/lang/rubytutorial/flowcontrol/

Brain, M., Chandler, N., & Crawford, S. (2021, April 8). *How Domain Name Servers Work | HowStuffWorks*. Computer | HowStuffWorks. Retrieved March 29, 2022, from https://computer.howstuffworks.com/dns.htm

Castello, J. (2019, April). *An Overview of Data Structures For Ruby Developers*. RubyGuides. Retrieved March 30, 2022, from https://www.rubyguides.com/2019/04/ruby-data-structures/

Castello, J. (2019, October). *How to Use Queues in Ruby*. RubyGuides. Retrieved March 30, 2022, from https://www.rubyguides.com/2019/10/ruby-queues/

Castello, J. (2020, May). *Ruby Hash - Definition, Examples & Methods: The Ultimate Guide*. RubyGuides. Retrieved March 30, 2022, from https://www.rubyguides.com/2020/05/ruby-hash-methods/

Christensson, P. (2006). *Tag Definition*. TechTerms. Retrieved March 28, 2022, from https://techterms.com/definition/tag

Christensson, P. (2011, June 1). *Home Technical Terms Markup Language Definition*. TechTerms. Retrieved March 27, 2022, from https://techterms.com/definition/markup_language

Cloudflare. (n.d.). *What is a packet? | Network packet definition*. Cloudflare. Retrieved March 28, 2022, from https://www.cloudflare.com/en-au/learning/network-layer/what-is-a-packet/

Cloudflare. (n.d.). *What is HTTP?* Cloudflare. Retrieved March 30, 2022, from https://www.cloudflare.com/en-au/learning/ddos/glossary/hypertext-transfer-protocol-http/

Cloudflare. (n.d.). *What is routing? | IP routing*. Cloudflare. Retrieved March 29, 2022, from https://www.cloudflare.com/en-au/learning/network-layer/what-is-routing/

Coding Mentors. (2018, June 4). *Compiler and Interpreter: Compiled Language vs Interpreted Programming Languages*. YouTube. Retrieved April 4, 2022, from https://www.youtube.com/watch?v=I1f45REi3k4

Datta, A. (2017, March 13). *Did artificial intelligence deny you credit?* The Conversation. Retrieved April 10, 2022, from https://theconversation.com/did-artificial-intelligence-deny-you-credit-73259

de Bruijn, T. (2018, September 24). *#to_s or #to_str? Explicitly casting vs. implicitly coercing types in Ruby*. AppSignal Blog. Retrieved April 5, 2022, from https://blog.appsignal.com/2018/09/25/explicitly-casting-vs-implicitly-coercing-types-in-ruby.html

Edspresso Team. (n.d.). *Data types in Ruby*. Educative.io. Retrieved April 6, 2022, from https://www.educative.io/edpresso/data-types-in-ruby

Everette, J. (2021, July 11). *Beginner's Guide to Uber*. Lifewire. Retrieved April 10, 2022, from https://www.lifewire.com/how-does-uber-work-3862752

Fox, P. (n.d.). *IP packets (article) | The Internet*. Khan Academy. Retrieved March 28, 2022, from https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:the-internet/xcae6f4a7ff015e7d:routing-with-redundancy/a/ip-packets

Fox, P. (2020). *IP addresses (article) | The Internet*. Khan Academy. Retrieved March 28, 2022, from https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:the-internet/xcae6f4a7ff015e7d:addressing-the-internet/a/ip-v4-v6-addresses

Fox, P. (2020). *Transmission Control Protocol (TCP) (article)*. Khan Academy. Retrieved March 30, 2022, from https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:the-internet/xcae6f4a7ff015e7d:transporting-packets/a/transmission-control-protocol--tcp freeCodeCamp. (2019, December 5). *The Advantages and Disadvantages of JavaScript*. freeCodeCamp. Retrieved April 4, 2022, from https://www.freecodecamp.org/news/the-advantages-and-disadvantages-of-javascript/

Garsiel, T., & Irish, P. (2011, August 5). *How Browsers Work: Behind the scenes of modern web browsers*. HTML5 Rocks. Retrieved March 30, 2022, from https://www.html5rocks.com/en/tutorials/internals/howbrowserswork/

Harnish, B. (2020, December 18). *What Is HTTPS: The Definitive Guide to How HTTPS Works*. SEMrush. Retrieved March 30, 2022, from https://www.semrush.com/blog/what-is-https

Hogan, B. (2017, October 6). *Understanding Data Types in Ruby*. DigitalOcean. Retrieved April 6, 2022, from https://www.digitalocean.com/community/tutorials/understanding-data-types-in-ruby

HowStuffWorks.com. (2021, March 30). *What is a network packet? | HowStuffWorks*. Computer | HowStuffWorks. Retrieved March 28, 2022, from https://computer.howstuffworks.com/question525.htm

Isaac, M. (2017, March 3). *How Uber Deceives the Authorities Worldwide*. The New York Times. Retrieved April 10, 2022, from https://www.nytimes.com/2017/03/03/technology/uber-greyball-program-evade-authoritie s.html

JavaScript Tutorial. (n.d.). *JavaScript Queue*. JavaScript Tutorial. Retrieved April 4, 2022, from https://www.javascripttutorial.net/javascript-queue/

Knowles, B. (2020, November 11). *The Basics Of Chrome DevTools. A Beginner's Guide | by Bryn Knowles | The Startup*. Medium. Retrieved March 30, 2022, from https://medium.com/swlh/the-basics-of-chrome-devtools-4d69a102a699

Kyrnin, J. (2021, March 21). *What Are Markup Languages? — Web Design*. ThoughtCo. Retrieved March 26, 2022, from https://www.thoughtco.com/what-are-markup-languages-3468655

Launch School. (n.d.). *Ruby Hashes - A Detailed Guide*. Launch School. Retrieved March 30, 2022, from https://launchschool.com/books/ruby/read/hashes

Launch School. (n.d.). *Ruby Hashes - A Detailed Guide*. Launch School. Retrieved March 30, 2022, from https://launchschool.com/books/ruby/read/hashes

Lutkevich, B. (2021, October). *What is Transmission Control Protocol (TCP)? Definition from SearchNetworking*. TechTarget. Retrieved March 30, 2022, from https://www.techtarget.com/searchnetworking/definition/TCP

MDN Contributors. (2021, October 7). *Type coercion - MDN Web Docs Glossary: Definitions of Web-related terms | MDN*. MDN Web Docs. Retrieved April 5, 2022, from https://developer.mozilla.org/en-US/docs/Glossary/Type_coercion

MDN Contributors. (2022, January 18). *Control flow - MDN Web Docs Glossary: Definitions of Web-related terms | MDN*. MDN Web Docs. Retrieved April 6, 2022, from https://developer.mozilla.org/en-US/docs/Glossary/Control_flow

Michael, K., McNamee, A., & Michael, M. G. (2006, July 27). *The emerging ethics of humancentric GPS tracking and monitoring* [Conference Paper]. Copenhagen, Denmark. https://ro.uow.edu.au/cgi/viewcontent.cgi?article=1384&context=infopapers

Nadalin, A. (2018, July 29). *WASEC: understanding the browser*. Alessandro Nadalin. Retrieved March 30, 2022, from https://odino.org/wasec-understanding-the-browser/

Pancake, C. M. (2018, August 8). *Programmers need ethics when designing the technologies that influence people's lives*. The Conversation. Retrieved April 10, 2022, from https://theconversation.com/programmers-need-ethics-when-designing-the-technologies-that-influence-peoples-lives-100802

Programiz. (n.d.). *Interpreter Vs Compiler : Differences Between Interpreter and Compiler*. Programiz. Retrieved April 4, 2022, from https://www.programiz.com/article/difference-compiler-interpreter

Reuters. (2017, May 4). *Uber faces criminal investigation after evading the law with 'Greyball' tool*. The Guardian. Retrieved April 10, 2022, from https://www.theguardian.com/technology/2017/may/04/uber-criminal-investigation-greyball

Ritter, S. (2020, March 31). *The Ethical Data Dilemma: Why Ethics Will Separate Data Privacy Leaders From Followers*. Forbes. Retrieved April 10, 2022, from https://www.forbes.com/sites/forbestechcouncil/2020/03/31/the-ethical-data-dilemma-why-ethics-will-separate-data-privacy-leaders-from-followers

Saini, A., Singh, A., & Rai, A. (2021, May 19). *Ruby | Arrays*. GeeksforGeeks. Retrieved March 30, 2022, from https://www.geeksforgeeks.org/ruby-arrays/

Simplilearn. (2021, October 28). *Top 10 Reasons to Learn JavaScript*. Simplilearn. Retrieved April 4, 2022, from https://www.simplilearn.com/reasons-to-learn-javascript-article

Smith, J. (2022, February 19). *Compiler Vs. Interpreter: What's the Difference?* Guru99. Retrieved April 4, 2022, from https://www.guru99.com/difference-compiler-vs-interpreter.html

Study Tonight. (n.d.). *Queue Data Structure*. Studytonight. Retrieved April 4, 2022, from https://www.studytonight.com/data-structures/queue-data-structure

Trackometer. (2021, November 24). *Is GPS Tracking legal?* GPS Trackers by trackOmeter. Retrieved April 10, 2022, from https://www.trackometer.net/news/is-gps-tracking-legal

Varagouli, E. (2021, October 18). *What Are Markup Languages & How Do They Work?* SEMrush. Retrieved March 26, 2022, from https://www.semrush.com/blog/markup-language/

Veeraraghavan, S. (2022, March 16). *12 Best Programming Languages to Learn in 2022*. Simplilearn. Retrieved April 4, 2022, from https://www.simplilearn.com/best-programming-languages-start-learning-today-article

Watts, J. (2021, February 9). *Is GPS Vehicle Tracking Legal for Australian Businesses?* Expert Market. Retrieved April 10, 2022, from https://www.expertmarket.com/au/vehicle-tracking/australian-law-on-vehicle-tracking