1. オブジェクト指向とは何かを述べてください
   a. 特徴 3 つ、また、その説明を含めてください
   b. 具体例を含めてください

【回答】
■オブジェクト指向とは

現在最も主流となっている、オブジェクト(モノ)中心に考えた
プログラミングスタイルのこと。

プログラミングを「モノの集まり」としてとらえており、
できあがったモノを組み合わせてサービスを作る開発手法です。

例えば、Twitter にはユーザー、フォロワー、つぶやきというモノがあります。
ユーザーやフォロワーには人という実体があるため理解しやすいですが、
オブジェクト指向では、「つぶやき」という実体のない概念のようなモノも
モノ(オブジェクト)として扱うことができます。

何をモノとして扱うか扱わないかは開発者が決定します。

■オブジェクト(モノ)の 3 つの特徴

モノは以下の 3 つの要素から成り立っています。
・クラス　　→プロパティとメソッドから成るモノの設計書
・プロパティ →そのモノの構成要素
・メソッド　→そのモノが行うふるまい

例えば、Twitter のユーザー(モノ)は、以下の要素で成り立っています。

ユーザーのプロパティ→「構成要素」
   ・ユーザー名
   ・メールアドレス
   ・パスワード
   ・登録日　など

ユーザーのメソッド→「ふるまい」

・会員登録をする
・つぶやく
・リツイートをする
・フォローする　など

上記の構成要素が集まってユーザーというモノを表しており(プロパティ)、
「ふるまい」はモノがどういう行動をとれるのかを表しています(メソッド)。
そして、構成要素とふるまいがまとめられた設計書をクラスと呼びます。

また実体のない「つぶやき」においても、ユーザーと同様にモノとして定義することができます。

つぶやきの構成要素

・つぶやき内容
・つぶやいた人の ID
・つぶやいた日　など

つぶやきのふるまい

・いいねされる
・返信される
・リツイートされる　など

2. Github flow とは何かを述べてください
a. 下記の文言を必ず含めてください
i. リポジトリ
ii. main
iii. リモート
iv. ブランチ

【回答】
■Github flow とは

Github の開発で使用されているワークフローのこと。

後述する手順で GitHub Flow を進めると、
誰がいつどこを編集したのか、細かい履歴まで含め、
開発中のファイルを保存管理することができます。

■GitHub Flow の手順
---
① リモートリポジトリ(データの格納場所)を作る
② クローンする（リポジトリのコピーを取得し自身の PC 上で編集できる状態にする）
③ テキストエディタでクローンしたファイルを修正する
④ ブラウザで確認する
⑤ 保存しておきたいタイミングで commit(保存)する
⑥ 3-5 を繰り返して開発を進める
⑦ 1 つの機能やコンテンツが完成したら push(リモートリポジトリに渡す)する

サーバー上にデータの格納場所を作り、自身の PC 上に同じものをコピーします。
PC 上でファイルを修正し、保存したいタイミングで commit し変更履歴を保存。
機能が完成したら push してサーバー上のリモートリポジトリにデータを渡す、
といった流れになっています。

それぞれの用語を解説します。

・リポジトリ
　データを保管しておく貯蔵庫のようなイメージ。
　Git にはリモートリポジトリとローカルリポジトリの 2 種類があります。

・リモートリポジトリ：サーバー上に 1 つだけあるリポジトリ
　開発者それぞれが編集したファイルの差分をまとめて、
　綺麗に統合したファイルを保存している場所です。

　各開発者が push してきたファイルをまとめて統合(merge)する役割があります。

・ローカルリポジトリ：各開発者の PC 上にあるリポジトリ
　手順②で自身の PC にクローンされたファイルの
　格納場所がローカルリポジトリです。

　ローカルリポジトリにファイルを修正して保存した履歴を貯めておくことが可能。
　修正して保存することを add、
　保存した履歴を貯めることを commit と呼びます。

■ブランチとは

個人で開発をする場合には前述した手順で開発を進めても問題ありませんが、
チームで開発を行う場合、各開発者のコードが
プロジェクト本体に影響を与えてしまう可能性があるため、
ブランチを切って(作って)開発を進めます。

ブランチとは、「コードを乗せたトラック」とイメージしてください。

GitHub では初めから main ブランチというブランチが用意されています。
コードを書いた後に commit(保存)し、push(リモートリポジトリに渡す)した場合、
main ブランチというトラックが、書いたコードを乗せて
GitHub サーバーに運ぶというイメージです。

チーム開発では header ブランチや footer ブランチなど、
自分の担当している箇所のブランチを切り、そこで作業を行います。
作業が完成したら main ブランチに合流(merge)します。

merge する前にチームメイトに、
このコードで問題がないか確認して merge を承認してもらう
Pull Request(プル リクエスト)という機能もあります。

問題なくリクエストが承認されれば、
作業したブランチが merge され、main ブランチに反映されます。

merge 後は最新のコードを反映させるために
main ブランチを Pull(自分の PC に反映)し、次の作業へ進みます。

3. サーバーサイドエンジニア・フロントエンジニアとはどのような違いがあるかを述べてください。

【回答】
■フロントエンジニア

Web サイトの見える部分を作るエンジニアのこと。

見える部分とは、Web サイトにアクセスした際に　画面に表示される見た目のことです。

「この文字をここに配置して、この画像をここに配置する」
といった Web サイトの見た目を作る部分を担当しています。

■サーバーサイドエンジニア

Web サイトの目に見えない部分を作るエンジニアのこと。

Web サイトは見た目を整えれば完成。
ではなく、目に見えない裏側の処理も作る必要があります。

例) EC サイトで商品を注文するときの処理
　　会員サイトにログインするときの処理

我々がどこかの会員制の Web サイトにログインした際、
会員情報が保存されているデータベースから
入力されたメールアドレスとパスワードが一致しているか確認し、
一致していたらログインさせる。
といった、目は見えない裏側の処理が実行されています。

このような裏側の処理を作る部分を担当しているのが、
サーバーサイドエンジニアです。

4. AWS とは何ですか。特徴を述べてください。

【回答】
■AWS とは

AWS とは Amazon Web Services の略で、
amazon が提供するクラウドコンピューティングサービスのことです。

クラウドコンピューティングサービスとは
インターネットを介して、サーバーやストレージ、データベースといった
コンピューターを使ったさまざまなサービスを利用することを指します。

PC とインターネットに接続できる環境さえあれば、
サーバーや大容量のストレージ、データベースなどを
必要な時に必要な分だけ利用することが可能です。

クラウドコンピューティングサービスが登場する前までは
自社の建物内などに物理サーバーを設置して運用を行う必要がありました。
この運用形態をオンプレミスと呼びます。

■AWS の特徴

・豊富なサービス
　100 以上のサービスがあり、
　必要に応じ機能を組み合わせて導入することで、
　さまざまなサービスを利用することができる。

例）仮想サーバーとデータベースの組み合わせなど

・コスト面
　初期費用なし。費用は使用した分だけ。
　使わないサービスに費用を払う必要はありません。

・セキュリティ
amazon が自社のサーバーとして利用していたものを
提供しているサービスなので、最新のセキュリティ状態を保っている。
各国の規制やコンプライアンスにも対応。

セキュリティ性が高いため、
金融業界や政府機関でも利用されています。

・パフォーマンス
クラウドサーバーは、
最新世代のハードウェアへ定期的にアップグレードされているため、
サーバーの質が常に高い。

AWS は上記のような特徴があり、
非常に優れたクラウドコンピューティングサービスです。
クラウド市場のシェア No.1 を誇り、企業の導入事例も非常に多くあります。

5. Docker とは具体的に何ができる技術ですか。また Docker を導入するメリットを述べてください。

【回答】
■Docker とは

コンテナ型の仮想環境を構築できる OSS(オープンソースソフトウェア)のこと。

仮想環境とは、1 つのハードウェアの中に仮想的な環境を構築したもの。
例えば、Mac OS がインストールされている Mac の PC 上で
Windows OS を仮想的に用意する、といったようなことを指します。

この場合、土台となっている Mac OS を「ホスト OS」
仮想環境上の Windows OS を「ゲスト OS」と呼びます。

仮想環境を構築することで、Mac 上で通常動かすことのできない
Windows OS や Linux OS を動かすことができるようになります。

サービス開発時において、Docker を使うことで
自身の PC(ローカル環境)に
実際にサービスを動かす環境を仮想的に構築することができるため、
本番環境と同様の環境で確認しながら、開発を行うことが可能になります。

■Docker のメリット

・Docker コンテナはホスト OS のカーネルを共有して使うので軽くて早い
　→仮想環境の種類にて詳しく解説

・Docker イメージがあれば、簡単に環境構築ができる
（コンテナに必要なファイルをまとめたもの）

・Dockerfile を使えば構築手順をファイルにまとめられる
（Docker イメージを自動構築する設定ファイル）
　→インフラをコード化できる

■仮想環境の種類

・コンテナ型

あるアプリに必要な環境を OS レベルでパッケージ化してまとめた箱のようなイメージ。
動作させる際はホスト OS を利用し、コンテナエンジン上で作動させている。

マシン -- ホスト OS -- コンテナエンジン --コンテナ［アプリケーション、ライブラリ］
--コンテナ［アプリケーション、ライブラリ］

ホスト OS のカーネル(OS の中核)を利用しているため、
すくないリソースで実行することが可能。
Docker はこのコンテナ型にあたります。

・ハイパーバイザー型

ホスト OS の上でゲスト OS を別で起動させる仕組みの仮想環境。

マシン -- ホスト OS -- ハイパーバイザ --[ゲスト OS、アプリケーション、ライブラリ]
--[ゲスト OS、アプリケーション、ライブラリ]

ホスト OS とゲスト OS が別で起動しているため、
コンテナ型と比較して、より多くの CPU やメモリを消費してしまう。